The Dissertation Committee for Tao Luo
certifies that this is the approved version of the following dissertation:

Nanometer VLSI Placement and Optimization for Multi-Objective Design Closure

Committee:

David Z. Pan, Supervisor

Jacob Abraham

David Newmark

Nur Touba

Michael Orshansky

John Zhang

Nanometer VLSI Placement and Optimization for Multi-Objective Design Closure

by

Tao Luo, M.S., B.S.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

December 2007

To my parents and my wife.

Acknowledgments

I would like to thank my advisor, first and foremost, Dr. David Pan, for all his guidance, encouragement, support, and patience throughout my Ph.D study and his assistance on my career planning. What I have learned from Dr. Pan have made a part of my life. I am very grateful for all that he has done for me.

Special Thanks go to Dr. David Newmark and Dr. Charles Alpert, who have provided me research opportunities in AMD North Austin Design Center and IBM Austin Research Lab. Their insights on research and mentoring experiences have made this dissertation possible. I believe what I have learned from them will continue guiding me beyond my graduation.

I am also extremely grateful to the members of my Ph.D. committee, Dr. Jacob Abraham, Dr. Nur Touba, Dr. Michael Orshansky, and Dr. John Zhang, for their interests in this research and supports during my completion of this dissertation.

Many thanks to other staff members in the Department of Electrical and Computer Engineering, especially Andrew Kieschnick and Melanie Gulick. Andrew is such a skillful and conscientious system administrator who has never failed to provide in time support to our research. Melanie is a warm-hearted graduate program coordinator that has been always so friendly and helpful to the students.

I am also grateful to students in UT VLSI Design Automation group for

many inspiriting discussions on research and their helps on varies aspects.

I would like to thank my family, especially to my understanding and patient wife, Hwa Fen, for supporting me in my Ph.D. pursuit. Without her support, I could not imagine my Ph.D. being accomplished.

Finally, I would like to express my appreciation to the donors of the Engineering Doctoral Fellowship for their generous financial support and the Semiconductor Research Corporation for funding my research with contract number 1321-001.

Nanometer VLSI Placement and Optimization for Multi-Objective Design Closure

Publication No. _____

Tao Luo, Ph.D.
The University of Texas at Austin, 2007

Supervisor: David Z. Pan

In a VLSI physical synthesis flow, placement directly defines the interconnection, which affects many other design objectives, such as timing, power consumption, congestion, and thermal issues. With the scaling of technology, the relative interconnect delay increases dramatically. As a result, placement has become a bottleneck in deep sub-micron physical synthesis. In this dissertation, I propose several optimization algorithms from global placement, placement migration, timing driven placements, to incremental power optimizations for multi-objective VLSI design closure. The first work is DPlace, a new global placement algorithm that scales well to the modern large-scale circuit placement problems. DPlace simulates the natural diffusion process to spread cells smoothly over the placement region, and uses both analytical and discrete techniques to improve the wire length. However, global placement is never sufficient for multi-objective design closure, a variety of design objectives have to be improved incrementally, such as timing, routing congestion, signal integrity, and heat distribution. Placement migration is a critical step

to address the cell overlaps appearing during incremental optimizations. To achieve high placement stability, I propose a computational geometry based placement migration flow to cope with placement changes, and a new stability metric to measure the "similarity" between two placements accurately. Our placement migration algorithm has clear advantage over conventional legalization algorithms such that the neighborhood characteristics of the original placement are preserved. For timing closure in high performance designs, I present a linear programming based incremental timing driven placement to improve the timing on critical paths directly. I further present an efficient timing driven placement algorithm (Pyramids). Two formulations of Pyramids are proposed, which are suitable for different optimization stages in a physical synthesis flow. Both approaches find the optimal location for timing of a cell in constant time, through computational geometry based approaches. For fast convergence of design closure, placement should be integrated with other optimization techniques. I propose to combine placement, gate sizing and Vt swapping techniques to reduce the total power consumption, especially the leakage power, which is becoming increasingly critical for nanometer VLSI design closure.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

For more than four decades, the technology in semiconductor industry has been advancing in the remarkable pace predicted by the Moore's law. However, since the feature size of the devices scales to nanometer era (90nm and below), many entangled physical effects in deep submiron region, such as the interconnect, leakage, and thermal, have made the design closure in physical design even more challenging.

1.1 Background of physical design and motivations

Physical design/synthesis is a complex multi-phases process. In modern VLSI design, physical design/synthesis refers to the key design implementation stages from register-transfer level (RTL) to GDSII layout, including floorplanning, logic synthesis, placement, gate sizing, buffer insertions, Vt swapping, routing, etc.

In the process of design closure, the global placer generates an initial placement to minimize the wire length. Once large electrical violations are corrected, a timing profile is obtained through the timing analysis. The timing profile is used as the basis of the subsequent timing driven global placement iterations. Related to the focus of this dissertation, a physical synthesis flow typically has the following

design phases [5, 6]:

1. Wire-length driven global placement, which generates an initial placement with minimized wire length.

2. Electrical correction, which uses gate sizing and buffering to fix large electrical violations to make the design in a reasonable shape.

3. Timing driven global placement. The timing information is incorporated in global placement to improve the timing of the design.

4. Timing driven detailed placement. The design is legalized, and a variety of incremental optimization techniques are employed to further improve the timing.

5. Critical path optimization. The design is largely in a good shape at this point. A variety of techniques, including placement, sizing, and buffering, are used to further improve the timing for the most critical paths.

6. Power optimization. To size down gates on non-critical part of the chip to recover some power.

Placement is one of the most important phases in a physical synthesis flow, which interacts with almost every physical synthesis phase. Briefly, placement is the step to assign standard cells and blocks into the chip region to satisfy certain design objectives, such that all standard cells being placed on circuit rows without overlap between any circuit component. As a result, interconnects are largely determined in the placement phase.

Figure 1.1 shows the scaling trend of the interconnect delay and gate delay predicted by *International Technology Roadmap for Semiconductors* (ITRS) [9]. As technology scales, the size of the transistors/gate decreases, and the speed of transistor increases. However, the interconnect delay decreases far less rapidly than the scaling of feature size, and the relative delay of global wire increases drastically, as shown in Figure 1.1. Therefore, the timing closure in physical design is much harder with the technology scaling.

Moreover, although a transistor consumes less dynamic power as it shrinks, the leakage current increases due to scaling. Nowadays, power has become one of the dominant performance liming constraints.

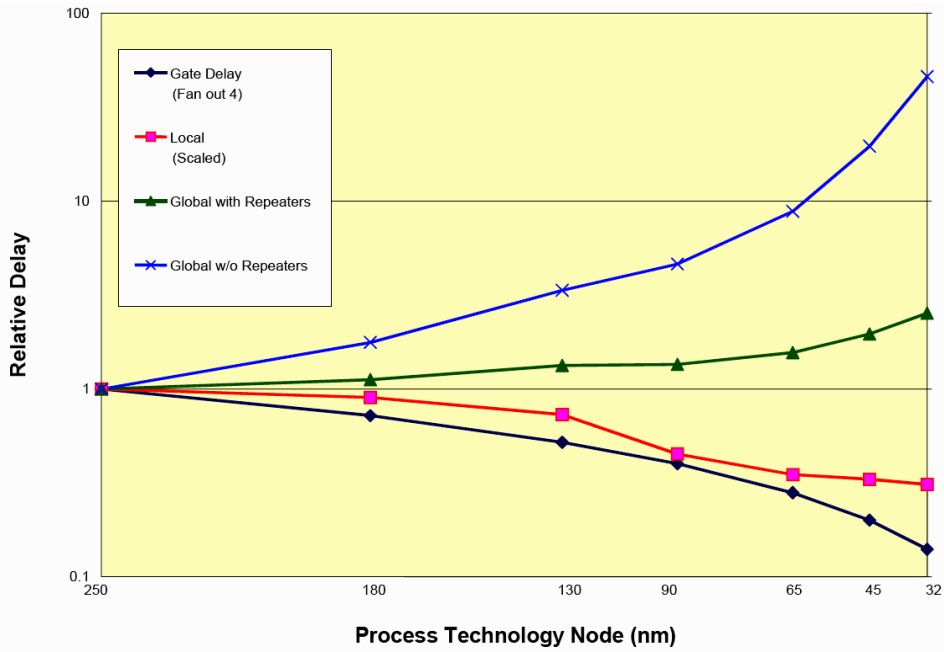


Figure 1.1: ITRS2005: Delay for metal 1 and global wiring versus feature size

As placement determines global interconnections, it affects almost every design objective, such as timing, signal integrity, routability, thermal, and power. The following subsection gives an overview about the role of placement in modern physical synthesis.

1.1.1 The role of placement in modern physical synthesis

Besides placement, among all other optimization techniques, gate sizing and Vt swapping are powerful for power reduction and timing improvement. Both gate sizing and Vt swapping interact with placement closely. Therefore, placement plays a pivotal role for VLSI design closure.

Major placement objectives include the total wire length, delay, congestion, power, thermal, the runtime, and stability of the placement algorithm. As the signal delay is quadratic to wire length, and power is proportional to the wire length, the total wire length minimization is the primary objective. The congestion is also very important, which determines the routability of the chip. Furthermore, the runtime of placement algorithm has become one of the main concerns. Nowadays, some industry placement problems have multi-million objects with a large amount of blockages [66, 65]. To generate one solution of a multi-million gates design may take tens of hours.

Placement is usually divided into two stages, the global placement and the detailed placement. Although there is no precise criteria to split the global and the detail placement, the global placement in general spreads cells roughly over the placement region with global wire length minimization, and the detailed placement

snaps cells into circuit rows and removes the residual overlap for a given global placement solution.

1.1.1.1 Global placement

Historically, existing global placement algorithms can be roughly classified into three major categories, i.e., the simulated annealing , iterative partitioning based approach, and analytical placement. Traditional objective in global placement is total wire length optimization. Total wire length is the sum of all interconnects among cells and most of existing placements use the Half Parameter Wire Length (HPWL) for wire length estimation.

Simulated annealing (SA) based approach formulates the placement as a metallic annealing process [62], which iteratively changes the current solution by a random small amount, evaluates the new solution, and adopts the new solution by a certain probability that controlled by parameters. For very small circuits, SA based approach achieves the best solutions. However, the runtime of the SA placement is prohibitive and unrealistic for for modern placement.

Min-cut or partitioning based placement recursively partitions cells into smaller chip subregions [16, 87, 91]. The objective for every partitioning step is to find a cell partition that minimizes the sum of the weighted cut sizes. Typical min-cut based placements include Capo [16] and Fengshui [1]. In the ISPD placement contests [45, 65], the wirelength results of partitioning based placements are not very competitive since the cut-size is not a direct wire length measurement.

Analytical placement has been successful in recent years and achieved im-

pressive results on wire length, scalability, and convergence. According to the results of ISPD 2005 and 2006 placement contest, most of the top ranked placers are analytical placers. The analytical placement models the placement objectives as a convex mathematical problem and solve the mathematical problem directly. Among analytical placement, quadratic placement optimizes the quadratic form of the HPWL [55, 32, 41, 50, 84, 86, 90] and log-sum-exp placement [47, 18] adopts a log-sum-exponential HPWL approximation patented by Naylor et al. [67].

1.1.1.2 Detailed and incremental placement

Given a global placement solution, detailed placement finds the legal position of cells and further improves the quality. Floating macros may occupy several rows, but there should be no overlap between objects. In detailed placement, wire length can be further reduced and various design objects can be improved, such as the routability, thermal, timing, etc. Detailed placement is more constrained than the global placement as most of cells are moved locally. Many detailed placers try to minimize the cell movements during the legalization.

Due to the complexity of modern nanometer designs, it is unlikely to design one placement algorithm that meets all design objectives in a single run. An industrial design flow may involve multiple improvement placement iterations. Given a legal placement, one may have to fix some design violations, insert some buffers, resize some gates, and re-legalize it. This is also referred as incremental placement problem. Incremental placement is similar as the detailed placement in the theme of starting from a placement to generate an improved legal solution of the original

placement. Timing driven placement focuses on improving the chip timing, and timing driven placement could be incremental, or at global and detailed placement levels.

1.1.2 Motivations

While CMOS scaling might be near its physical limit [9], the design technology is by no means close to its optimality. Placement problem is well known to be NP-hard. According to the experiments on a set of synthetic benchmarks with known optimal wire length, all existing placements are still far from optimal [30]. Although has been studied for decades, placement remains a bottleneck of VLSI design and continuously attracts research attentions.

Placement algorithms are predominantly wire length driven. However, the relatively increasing interconnect delay makes the problem even more challenging, especially for the timing driven placement.

Timing has been traditionally the most important design constraint. Meanwhile, chip designers nowadays are facing more and more power problems. The relative leakage power consumption increases quickly with the CMOS scaling. The more transistors on a chip, the more power is wasted on leakage, which leads to thermal issues. Traditional techniques for power reduction, such as clock gating, may not be sufficient if used alone. A rule of thumb in power optimization is that, the earlier to take power into consideration, the better chance to reduce the total power consumption. One has to combine power reduction techniques systematically, including the power aware placement, to conquer the rising obstacle of power

consumption.

The primary focus of this dissertation is to further push the limit of placement driven optimizations for multi-objective design closure. A set of algorithms at different phases in a physical synthesis flow have been developed, from global placement, placement migration, incremental timing driven placement to incremental power optimizations. Major design objectives, such as the wirelength, timing, and power are addressed from different perspectives.

The following section gives an overview of the dissertation.

1.2 Overview of the dissertation

This dissertation has studied five topics under the unified theme of placement and optimization for multi-objective design closure. Global placement is the foundation of all optimization phases in a physical synthesis flow. First, I propose DPlace in chapter 2, a novel global placement algorithm that is guided by a smooth cell spreading pre-placement stage to ensure the placement stability. Meanwhile, DPlace has unique properties to scale well with large-scale circuit placement problems. The DPlace framework interleaves diffusion spreading technique with analytical and discrete wire length minimization techniques to improve the wire length. The DPlace framework is flexible. Current implementation adopts the quadratic placement formulation for wire length minimization. However, any smooth spreading technique and effective analytical wire length minimization technique can be plugged in the two steps general placement framework in DPlace in chapter 2.

Despite the best effort, the global placement alone will never be sufficient close the timing in a physical synthesis flow. One cannot obtain an complete picture of the current timing until the placement has stabilized, and even timing-driven global placement will never be a complete solution. Thus, incremental placement is desired to take the existing placement and improve the timing or power characteristics of the design incrementally. In other words, incremental placement techniques help to stabilize the design and to evolve the design toward timing and power closure.

In incremental placements, the stability of the placement algorithm has high priority. In other words, the new placement solution generated by an incremental placer should be as close to the previous solution as possible. A large disturbance to the existing placement may destroy all previous optimization efforts [75]. In chapter 3, a computational geometry based placement migration tool is proposed to cope with design changes. The new migration tool perturbs a given illegal placement smoothly to remove cell overlap, meanwhile preserves the essential characteristics of the original placement, such as cell ordering, wire length, timing, etc.

While the placement migration algorithm implicitly improves the timing during optimizations, I present two additional timing driven placement works to explicitly improve the critical paths. In chapter 4, a linear programming (LP) based timing driven placement framework is proposed to explicitly improve the critical timing path in high performance design. The LP placement framework proposed is a hybrid approach, which combines the flexibility of the net based approach and the accurate timing view of the path based approach. Furthermore, the Pyramids

algorithm proposed in chapter 5 is an efficient timing driven placement algorithm suitable for both the global and detailed timing driven placement phrases. Given a movable cell, Pyramids algorithm finds the optimal locations of the cell in constant time to minimize the associated slacks.

A recent research has tried to reduce the power during placement phase. The power-aware placement work for dynamic power reduction is proposed in [23]. However, leakage power is not considered. To address the increasingly leakage issues, in chapter 6, we combine placement, gate sizing, and multiple-Vt assignment algorithm and methodology for total power optimization through effective slack distribution management.

Finally, we conclude in chapter 7.

Chapter 2

DPlace: A Stable and Efficient Global Placement

2.1 Introduction

Although circuit placement has been studied for decades, it continuously attracts research attentions. The placement problems grow rapidly in both problem size and complexity. Some industry placement problems contain multi-million gates and excessive number of blockages [66, 65]. In this chapter, we introduce a new diffusion based placer that scales well to large scale placement problems.

Historically, existing circuit placement algorithms can be roughly classified into three major categories, the simulated annealing [62], iterative partitioning based approach [16, 87, 91], and analytical placement approach [55, 32, 41, 84, 86, 18, 90, 21, 2]. In placement, the Half Parameter Wire Length (HPWL) is a common estimation of the routed wire length. Since HPWL model is not smooth and derivable, quadratic placement optimizes the quadratic form of HPWL [55, 32, 41, 84, 86, 90], and nonlinear model placement [48, 18, 21, 2] adopts a nonlinear estimation of HPWL model, typically the log-sum-exponential wire length approximation patented by Naylor et al. [67].

Four existing academic placers [48, 18, 21, 2] that use the log-sum-exponential wire model have achieved excellent wire length results. It is widely agreed that

placement uses log-sum-exponential wire model approximates the HPWL much closer than the quadratic estimation. However, although still controversial, some researchers believe that the quadratic placement potentially has advantages for timing driven placement, as the quadratic approximation of the HPWL gives larger penalty on longer wires. Meanwhile, the quadratic wire model is not restricted by any patent.

Diffusion is the flow of particles from a region of higher concentration to a region with lower concentration, until the concentration on both regions are equal. The cell spreading in placement shares similar philosophy as the natural diffusion process, where cells are driven from high density areas to low density areas. Diffusion based technique has been successfully applied to incremental placement optimization [75], and here we use diffusion to move cells in global placement.

In this chapter, we present DPlace: a stable and efficient diffusion based analytical placement. The DPlace starts with a seed placement, and interleaves two major optimization steps iteratively, 1) the diffusion based cell spreading step to even out the density distribution, 2) the wire length improvement step, which adopts both analytical and discrete wire length improvement techniques. The initial work of DPlace is presented in a technical report [59]. The following are a few characteristics of our approach, which differentiates our approach from previous analytical placement works.

- We propose a new placement framework based on, but not limited to, diffusion spreading and quadratic placement. Any smooth spreading and wire

length optimization technique can be inserted in our framework. Most importantly, we show that it is possible to deal with the overlap removing and wire length improvement tasks separately and achieve fairly good results. Such an approach provides flexibility to plug in additional optimizations/constraints that are non-trivial to integrate in conventional analytical placement framework.

- The *anchor cell* concept is introduced as the bridge integrating different optimization techniques for a stable and efficient global placement, which also significantly reduces the complexity of large scale placement. In each row of the hessian of the quadratic formulation, the anchors inserted change the number of non-zero entries of each row from the number of pins on the corresponding net to the number of pins on the corresponding cell. Such a change helps to speed up the linear system solver by 24 times for ISPD2005 benchmark suite.

- In our placement, to improve the wire length of a given placement by using quadratic formulation, we present a net weight linearization strategy that transforms the star model [63, 84] based quadratic objective into HPWL objective exactly.

In the following, we introduce the analytical placement models in section 2.3. The details of our global placement are described in section 2.4. Section 2.5 is about the legalization and detailed placement. We give the overall algorithm in

section 2.6 and show the experimental results in section 6.6, which followed by the summary in section 2.8.

2.2 Background: Force directed placement

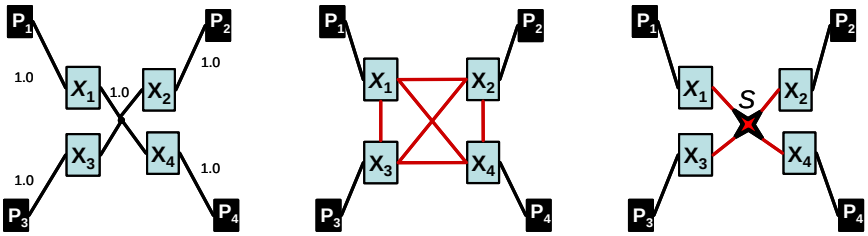
Among existing placement works, analytical placement has been successful in recent years and achieved impressive results on wire length, scalability and the speed of convergence. According to the reported results of ISPD 2005 and 2006 placement contest [45, 65], most of the top ranked placers are analytical placers.

Most of existing analytical placements are force directed. A typical force directed placement need to generate an initial placement with minimized total wire length without considering the cell overlapping constraint, and the initial placement solution has excessive overlap among cells. To push cells away from congestion, in subsequent iterations, a force directed placer adds "spreading force" or density constraints into the original wire length formulation. In force directed quadratic placement, the density constraints are combined into the optimization objective either by adding the spreading forces as constant force terms or by adding fixed points to implement the spreading forces. The following section gives an overview of the force-directed quadratic placement and the analysis of the essential concept in some of the existing force directed quadratic placement approaches.

2.2.1 Quadratic placement

In circuit placement, a netlist is normally modeled as a hyper-graph with each node representing an object/cell and each edge representing a net. Let x_i and

y_i denote the coordinates of each cell, Half Parameter Wire Length (HPWL) is used as an estimation of the routed wire length. Because the equation of HPWL is difficult to optimize mathematically, quadratic placement minimizes the square of the length and width of the bounding box of a net, commonly referred as the quadratic wire length.



(a) A circuit with 5 nets, 4 cells, and 4 pins, one of the nets is a 4-pin net

(b) Clique model transformation

(c) Star model transformation

Figure 2.1: Transformations of the multi-pin net into multiple two-in nets. Only the x coordinates are showed in these figures

In conventional quadratic placement [55], each multi-pin net is transformed into multiple two pin connections with proper weights by either the clique model in Figure 2.1(b), or the star model in Figure 2.1(c) [63, 84]. Clique model may increase the number of non-zero entries in the connectivity matrix significantly, as the example in Figure 2.7, which may slow down the quadratic solver. One k-pin net will be transformed into k connections in star model, as shown in Figure 2.1(c). The combination of the clique and star transformation is also referred as the hybid model [84].

For a two pin net $e_{i,j}$ that connects cell i and j, the quadratic wire length is

defined as $w_{i,j}((x_i - x_j)^2 + (y_i - y_j)^2)$, where $w_{i,j}$ denotes the weight of net $e_{i,j}$. The quadratic placement minimizes the sum of all quadratic wire length in the circuit. The optimization problems in x and y direction are separable and can be treated independently. Therefore, the cost function in x direction is given by

$$\Phi(x) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x} + const \tag{2.1}$$

Assuming there are n movable objects in the netlist. Let $\mathbf{A}$ denote the Hessian matrix of the quadratic system, which is essentially the n by n connectivity matrix of the netlist. $\mathbf{A}$ is symmetric and positive definite. $\mathbf{x}$ denotes the vector of x coordinates of all cells. $\mathbf{b}$ is the vector encoding all connectivity information between movable and fixed objects, and the pin offsets are captured in $\mathbf{b}$ as well. The minimizer of the cost function (2.1) can be obtained by taking the gradient of the cost function to zero, $\partial(\Phi(x))/\partial x = \mathbf{0}$, which is determined by the following system of linear equations

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{2.2}$$

Figure 2.2 shows a simple circuit with 2 movable cells and two fixed pins. The number associated with each net is the net weight. Cell 1 and 2 are in the force equilibrium status in Figure 2.2, i.e. the sum of the weighted quadratic wire length is the minimum.

2.2.2 Force-directed quadratic placement

Solving the unconstrained minimization problem in Equation (2.1) results a placement with significant overlap among cells. A placer needs to push cells

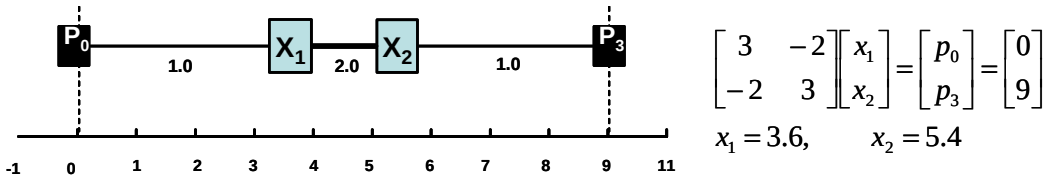


Figure 2.2: The quadratic placement formulation of a simple circuit in the x direction. p_0 and p_3 are the x coordinate of the fixed pins

around to remove overlap. Some placers recursively partition the placement region to spread cells, such as Gordian [55]. The force-directed placers add spreading forces into the system in each solving process and reduce the overlap iteratively. Figure 2.3 shows that cell 1 and 2 are too close to each other, a force directed placer adds forces to push cells away from the center.

To apply spreading forces into the optimization framework, there are mainly two types of strategy to implement the force, the *constant force addition* and the *fixed point addition* approach. In each placement iteration, Kraftwork [32] and FDP [86] add a constant force vector **f** to the right hand side of Equation (2.2). The fixed point based approach adds artificial pins and nets to move cells. mFar [41] uses multiple fixed virtual pins for each cell in every iteration, one is used to maintain a cell's force equilibrium state, and others are applied to perturb the cell. FastPlace [84] uses one fixed virtual pin for both purposes.

2.2.2.1 Constant forces

In every iteration, the force for each cell is computed to reduce the overlap. In constant force based approach, the force vector **f** is added to vector **b** in equation

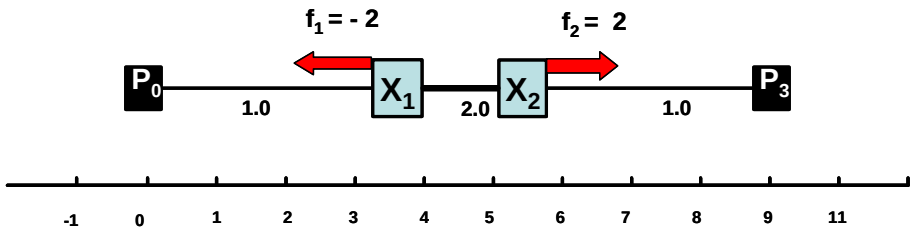


Figure 2.3: Force directed placement: adding forces to push cells out of the region with congestion

2.2. The solution of the modified quadratic system generates a placement with less overlap among cells. In the ith iteration, the force vectors used in 1 to $i-1$th iterations are accumulated to prevent cells collapsing back. The modified equation with constant forces is given by

$$\mathbf{Ax} = \mathbf{b} + \sum_{k=1}^{i-1} \mathbf{f}^k + \mathbf{f}^i \tag{2.3}$$

In constant force based approach, the Hessian (connectivity matrix) is not changed in each iteration unless the net re-weighting is involved. In such case, the Hessian **A** only needs to be pre-conditioned once in the beginning, which will save runtime as the matrix pre-conditioning is runtime expensive.

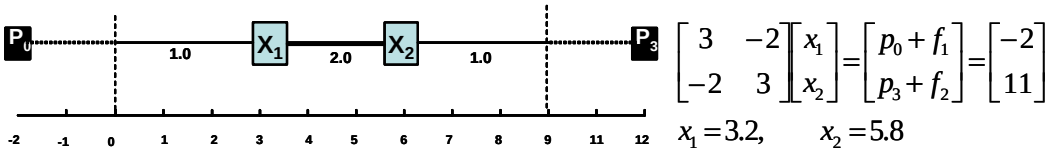
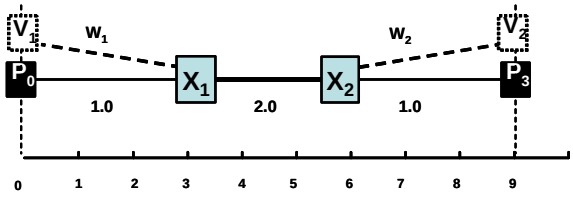


Figure 2.4: Adding the constant force on a cell is equivalent to shifting its connected objects

The physical meaning of adding a spreading force to one cell is equivalent to shifting its connected pins and cells. To add the spreading force in Figure 2.3, a

force vector is added into constant vector **b** in equation 2.2. To add a force vector is equivalent to shifting the connected objects of each cell, as shown in Figure 2.4. Pins are shifted outside of the chip, and cells may "jump" out the chip region if the magnitude and direction of the spreading forces are not properly adjusted. This tends to happen in the earlier placement iterations, where spreading forces are large and the force directions are not evenly distributed. Although such a scenario is not common in ISPD 2005 and 2006 benchmarks, where the initial density distributions are more even due to a large amount of fixed macros, the force scaling is tricky for placement with all movable objects, such as the ISPD02 benchmarks [43].

Because the connectivity matrix is not strictly diagonal dominant, and often ill-conditioned, the solver of the linear system may have stability problem [86], i.e. cells may jump around when large forces are added. FDP adds a small weight to a portion of the diagonal terms of the Hessian and the new Kraftwerk [81] adds weight to all diagonal terms. Such a strategy is equivalent to adding a virtual fixed pin and net to a cell, as shown in Figure 2.5, which affects the quadratic objective and improves the stability of the quadratic solver.



$$\begin{bmatrix} 3+w_1 & -2 \\ -2 & 3+w_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} p_0 + w_1 v_1 \\ p_3 + w_2 v_2 \end{bmatrix}$$

Figure 2.5: An example of the fixed point addition formulation, p_0, p_3, v_1, and v_2 are the x coordinate of the fixed real and virtual pins, respectively

2.2.2.2 Fixed point forces

In fixed-point methods, the fixed points and nets are added to the original system of linear equations to perturb the placement. In fixed-point based methods, adding a virtual fixed-point connection to a cell will add a diagonal term in the corresponding entry of the cell in the Hessian matrix **A** and the term in the constant vector **b**. In Figure 2.5, to add force to each cell, a virtual pin and connection are added to each cell with proper weight, and we can see the change in the Hessian and the constant vector in the figure. Therefore, adding a cell will make the corresponding row and column strictly diagonal dominant in Hessian **A**, and improve the condition number of the matrix. As a result, the fixed-point addition based method tends to be more stable.

The fixed point addition method guarantees cells moving inside the convex hull defined by the fixed points. If using a large weight for the virtual nets, cells have less mobility and tend to move steadily toward force directions. However, the added large virtual net weights may dominate the actual net connections and affect the optimization objective. On the contrary, if using very small virtual net weights, fixed-points will be off chip and cells may start to jump out of the boundary. In other words, the fixed point placement starts to behave similar as the constant force addition based method. Furthermore, in fixed point based approach, the connectivity weights will be updated in every iteration and the matrix needs to pre-conditioned in every solving iteration.

DPlace uses a completely different strategy to spread cells, as introduced in the following section.

2.3 Analytical placement model in DPlace

As mentioned earlier, two major tasks in a typical analytical placement iteration are to remove cell overlaps and to reduce the wire length. The conventional analytical placement formulates the wire length and density constraints into a mathematical problem. However, it is possible to address one issue at a time, which gives the flexibility to integrate additional optimizations or constraints during the global placement. In the proposed approach, we iteratively use diffusion to spread out cells, and repair the wire length afterwards.

2.3.1 Diffusion spreading

In general, an analytical placement tool starts with an initial placement with good wire length. Such an initial placement solution has excessive overlap among cells. Force directed placer adds "spreading force" or density constraints into the original wire length formulation to perturb the placement.

As diffusion has the advantage of smoothness in cell spreading and it preserves the cell relative ordering naturally, we use diffusion to spread cell out directly. Unfortunately, wire length objective is not modeled in the diffusion formulation. We then apply wire length reduction technique on the diffusion solution, such as the quadratic placement formulation, while preserving the diffusion improved density distribution.

In our approach, the quadratic solver is used to recover wire length increased during the diffusion spreading.

2.3.2 The proposed approach

We propose to interleave diffusion with analytical and discrete wire length minimization technique to improve the wire length, and the quadratic placement formulation is used. Meanwhile, any smooth spreading technique and analytical placement technique can be plugged in our two steps general placement framework.

The anchor cells are used to bridge the overlap reduction and the wire length reduction steps. We use the nets connecting anchor cells and real cells to formulate an unconstrained wire length minimization problem. Those nets are all "real" nets and we do not need to use any artificial nets in our formulation.

If the placement stability is not considered, once a netlist is changed, the placement solutions before and after the change could be completely different. In our approach, we may add additional constraints for placement stability. For example, we may "force" a placement to evolve in a desired way, which potentially provides flexibility for ECO placement.

2.4 Global Placement in DPlace

Similar as other analytical placements, our placer starts with a seed placement, which has a fair good initial wire length. In the beginning of each iteration, the diffusion algorithm is called to flow cells away from congested areas.

2.4.1 Diffusion based placement

Diffusion in placement is driven by the density gradient, i.e. the steepness of the density difference. Mathematically, the diffusion process is characterized by the following differential equation.

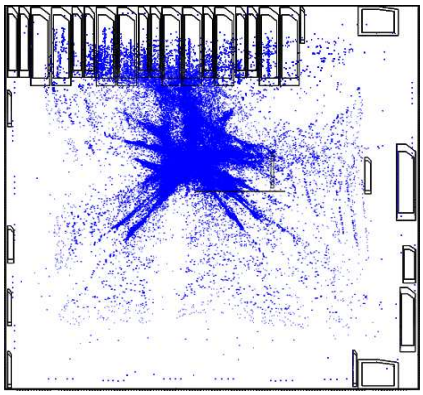
$$\frac{\partial d_{x,y}(t)}{\partial t} = D\nabla^2 d_{x,y}(t) \tag{2.4}$$

In the context of placement, $d_{x,y}(t)$ is the cell density at position (x,y) at time t. D is the diffusivity constant, which determines the speed of the diffusion process. The discrete approximation method in [75] can be used to solve the diffusion equation.

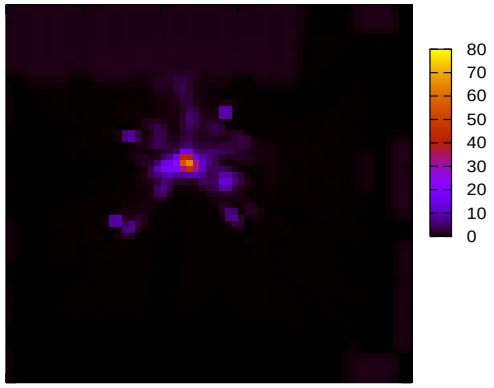
In diffusion based placement, the placement region is cut into equal size bins. The bin density is computed as the total cell area enclosed in the bin divided the bin area. The discrete solver we use to solve the diffusion equation evens out the densities between neighboring bins as time proceeds.

In every global placement iteration, cells are diffused from high density area to low density area. The diffusion based pre-placement takes k substeps, where k is relatively small in earlier placement iterations and becomes larger in the later iterations.

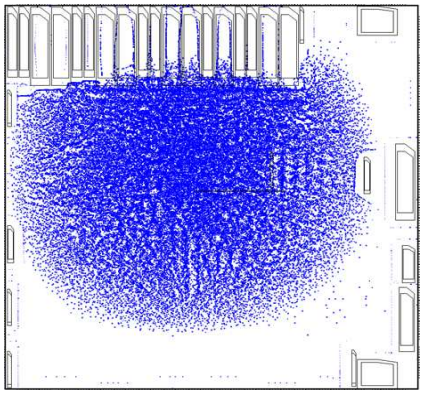
Figure 2.6 shows the diffusion based placement in circuit bigblue1 from ISPD2005 placement benchmark suite. The bin size used is 64x64. Figure 2.6(a) plots the initial placement seed, which is generated by the quadratic placement. Not surprisingly, the initial placement is extremely poor on density distribution.



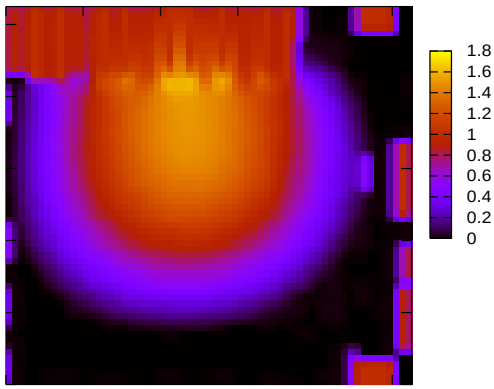
(a) Initial placement. HPWL: 6.84×10^7



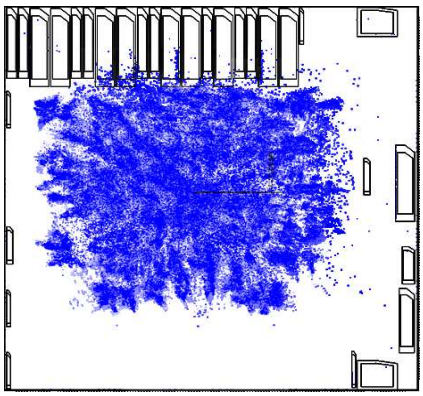
(b) Initial density map



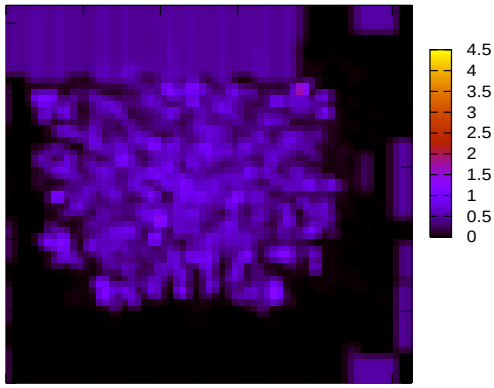
(c) After diffusion placement. HPWL: 17.3×10^7



(d) Smooth density distribution. However, the center is still congested



(e) After wire length reduction. HPWL: 9.63×10^6



(f) Better density distribution in a global view

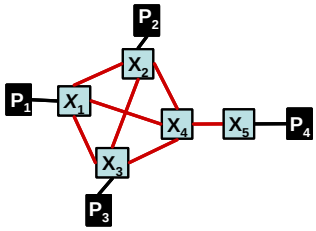
Figure 2.6: The diffusion and wire length reduction iteration in bigblue1(from ISPD 2005 placement benchmark suite)

The highest density is 80 times of the bin capacity and cells are highly congested in the middle of the placement region. Once applying a few iterations of the diffusion spreading, in Figure 2.6(c), we see how smoothly cells are moved and how effectively the density distribution is improved. The highest density in the placement region is reduced from 80 to less than 2 times of the bin capacity. Unfortunately, the wire length of the diffusion solution increases from 6.84 to 17.3. Figure 2.6(e) shows that once the wire length reduction techniques are applied, the wire length is reduced to 9.63. Although the densities in a few bins increase, we have a better density distribution in a global view.

The discrete diffusion algorithm is applied on a hierarchical bin structure. Our experiments suggest that a fixed bin size from the beginning to the end do not work well in global placement. If a small bin size is used, cells will spread smoothly, but slowly, which affects the convergence of the algorithm. Furthermore, as shown in Figure 2.6(e), it is difficult to reduce the density in the center area of congestion, if the bin size is too small. We use a large bin size in the beginning of the placement and reduce the bin size down gradually to resolve the local congestions. In addition, we adjust the density gradients according to local bin density distribution to improve the speed of convergence. i.e. let cells in highly congested area move faster.

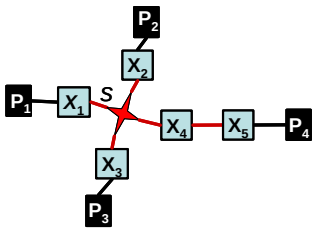
2.4.2 Anchor cells

To preserve the diffusion improved density distribution in the wire length repairing step, there are several choices to prevent cells collapsing back. For exam-



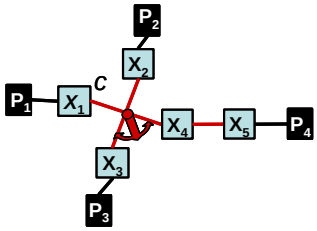
$$\begin{bmatrix} 1.75 & -0.25 & -0.25 & -0.25 & \\ -0.25 & 1.75 & -0.25 & -0.25 & \\ -0.25 & -0.25 & 1.75 & -0.25 & \\ -0.25 & -0.25 & -0.25 & 1.75 & -1 \\ & & & -1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ \\ p_4 \end{bmatrix}$$

Figure 2.7: The quadratic placement formulation by using clique model. For simplicity, we assume the weight of each transformed two-pin net is 0.25.



$$\begin{bmatrix} 1.25 & & & & & -0.25 \\ & 1.25 & & & & -0.25 \\ & & 1.25 & & & -0.25 \\ & & & 1.25 & -1 & -0.25 \\ & & & -1 & 2 & \\ -0.25 & -0.25 & -0.25 & -0.25 & & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ s \end{bmatrix} = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ \\ \end{bmatrix}$$

Figure 2.8: The quadratic placement formulation by using star model. S is the x coordinate of the star, which is a moveable object in the placement. For simplicity, we assume the weight of each transformed two-pin net is 0.25. The dimension of the Hessian matrix **A** is equal to the number of cells plus the number of stars



$$\begin{bmatrix} 1.25 & & & & \\ & 1.25 & & & \\ & & 1.25 & & \\ & & & 1.25 & -1 \\ & & & -1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} p_1 + 0.25c \\ p_2 + 0.25c \\ p_3 + 0.25c \\ \\ p_4 \end{bmatrix}$$

Figure 2.9: The quadratic placement formulation after the anchor cell insertion. C is the x coordinate of the anchor cell, which is a constant. The new Hessian matrix **A** is extremely sparse compared with that by using the star or clique formulation.

ple, we can fix a small percentage of cells, or attach some virtual cells to restrict the movement of real cells. And then, the quadratic engine is used to pull free cells toward a better location for improved wire length. In above scenarios, the fixed real or virtual cells are used as anchors to control the movement of real cells, and we name them "anchor cells".

We can either use one anchor per cell or one anchor per net in our framework. An efficient way is to use the star model to transform a portion of multi-pin nets into two-pin connections and use the star as the anchor of real cells. Compared with the method to use one anchor per cell, using stars as anchors will have less impact to the original wire objective and imposes less constraint on cell movements. And we can apply additional HPWL linearization technique by attaching anchor cells to the nets, as shown in later sections.

In the hybrid model based wire length transformation, multi-pin nets are converted into star and clique model. All stars will be added back into the Hessian matrix **A** as moveable objects, which may increase the dimension of the matrix significantly. In ISPD 2005 benchmark, by using star model with a pin threshold as 5 will increase the dimension of the matrix up to 40 percent. Under conventional formulation, solving one iteration of the system of linear equations with a dimension over 2 million will take several minutes.

Figure 2.7 shows the quadratic placement formulation of a toy circuit using the clique model. We see how dense is the hessian by using clique transformation. Figure 2.8 is the formulation by using the star model. The dimension of the Hessian will increase. Unlike stars, anchor cells are fixed objects. Therefore, anchor cells

will not increase the dimension of the Hessian **A**. Most importantly, in the anchor cell based quadratic formulation in Figure 2.9, we see that the Hessian matrix is extremely sparse compared with that by using either the star or clique models.

Let $\mathbf{A}'$ denotes the Hessian matrix in our new formulation. Anchor cells are not movable objects, thus do not appear in $\mathbf{A}'$. Matrix $\mathbf{A}'$ has the dimension as the number of movable objects in the netlist. Once cells are diffused, anchor cells are inserted at the gravity centers of their connected cells and locked. In such a way, anchor cells will pull free cells around in the subsequent wire length minimization.

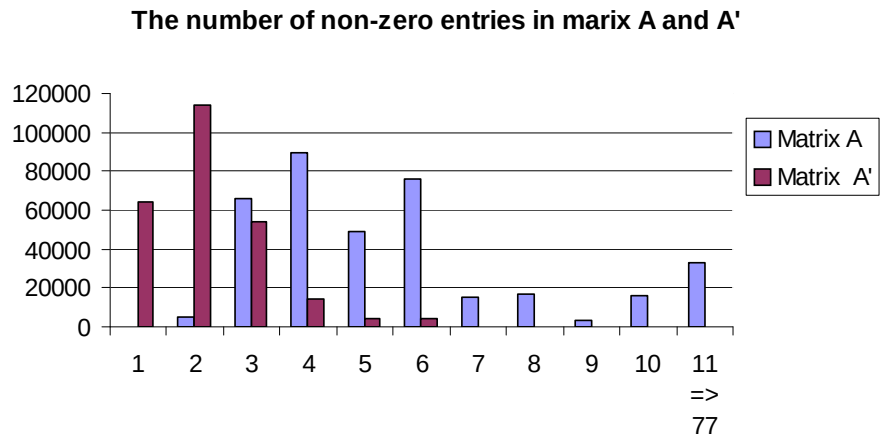


Figure 2.10: The comparison of non-zero entries in all rows in the sparse matrix **A** and $\mathbf{A}'$. The x-axis is the number of non-zero entries, the y-axis is the row counts. In hessian of the quadratic formulation, each row corresponds to the connected movable objects of each movable cell in the netlist. The insertion of anchor cells change the number of non-zero entries in each row from the pin degree of the corresponding net to the pin degree of the corresponding cell. Therefore, most of rows in matrix $\mathbf{A}'$ has only 2-3 non-zero entries. Such linear system takes very short time to solve.

Figure 2.10 shows the statistics of the number of non-zero entries in old Hessian **A** and new Hessian $\mathbf{A}'$ for circuit *adaptec*2 in ISPD 2005 benchmark. The

dimension of the Hessian $\mathbf{A}$ is 354K, while only 254K for the new Hessian $\mathbf{A}'$. In most of rows, the number of non-zero entries in $\mathbf{A}$ are around 3-6, and 1-2 in new Hessian $\mathbf{A}'$. Circuit *bigblue*4 in ISPD 2005 benchmark contains 2 million objects. By using the quadratic solver in [73], in our experiments for bigblue4, it takes 200 seconds for pre-conditioning and 75 seconds for solving using the conventional quadratic formulation, while only 11 seconds for preconditioning and 4 seconds for solving using our anchor cells based formulation.

Note that the anchor cell is different from the fixed point used in mFar [41] and FastPlace [84]. Anchor cell is the bridge connecting the overlap removing and wire length improvement stages. Fixed point is used to add the spreading forces back to the quadratic system and perturb the exiting placement.

2.4.3 HPWL transformation in a quadratic system

A major weakness of the quadratic wire formulation is that the quadratic objective is an approximation of HPWL for a two pin nets. Transforming a multi-pin net into multiple two-pin nets may enlarge the gap between HPWL and the actual objective to optimize. To alleviate such a problem, existing techniques iteratively linearize the quadratic wire length objective [80][81]. Here we propose a new liearization technique to transform the quadratic objective into HPWL exactly in our framework, which helps to reduce the gap between quadratic wire length and HPWL.

Assuming net e is connected with n cells, and HPWL in direction y is L_e. s is added to decompose the net e into n two-pin connections. Let l_i denotes the distance

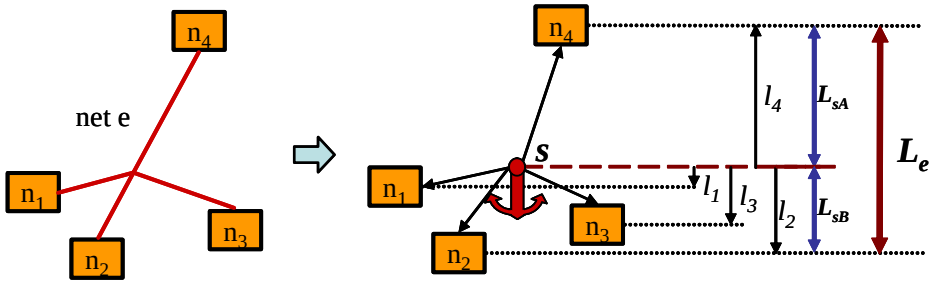


Figure 2.11: Net weights computation. $A = \{n_4\}, B = \{n_1, n_2, n_3\}$ in this example

between s and cell i and let w_i denote the weight of each two pin connection. We assign all cells into two sets based on if the cell n_i has a y coordinate large than that of star s. As a result, we have two sets, set $A = \{n_i : y_i > y_s\}$ and set $B = \{n_i : y_i < y_s\}$ for each star model transformation. We define the weight of each two pin net as follows.

$$
\begin{aligned}
w_i &= \frac{L_{sA}}{S_{AB} \times |y_i - y_s|}, \forall n_i \in A \\
w_i &= \frac{L_{sB}}{S_{AB} \times |y_i - y_s|}, \forall n_i \in B \\
&where \\
S_{AB} &= 0.5 \sum_{n_i} |y_i - y_s| \\
L_{sA} &= max\{y_i\} - y_s \\
L_{sB} &= y_s - max\{y_i\} \\
L_e &= L_{sA} + L_{sB}
\end{aligned} \tag{2.5}
$$

The anchor cell s is placed at the gravity center of all cells on net e, and S_{AB} is defined as the half of the sum of all distances from cell i to the star. Star s splits

the length L_e into two parts, L_{sA} and L_{sB}, as shown in Figure 2.11.

In the following, we show that the above net weighting strategy transforms the quadratic wire length objective into HPWL objective exactly.

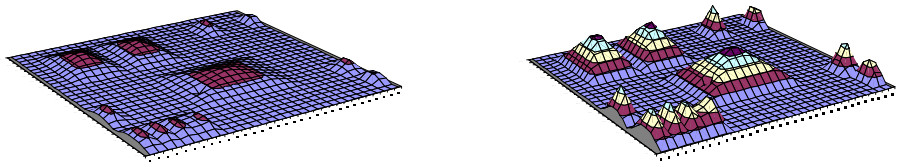
$$\begin{aligned}
\sum_{i=1}^{n} w_i(y_i - y_s)^2 &= \sum_{i\in A} \frac{(y_i - y_s)^2 \times L_{sA}}{|y_i - y_s| \times S_{AB}} + \sum_{i\in B} \frac{(y_i - y_s)^2 \times L_{sB}}{|y_i - y_s| \times S_{AB}} \\
&= \frac{L_{sA}}{S_{AB}} \sum_{i\in A} |y_i - y_s| + \frac{L_{sB}}{S_{AB}} \sum_{i\in B} |y_i - y_s| \\
&= L_{sA} + L_{sB} = L_e
\end{aligned} \tag{2.6}$$

Figure 2.11 shows an example of 4-pin nets transformation.

2.4.4 Fixed blockages

Fixed blockages are obstacles to cell spreading. Modern designs may contain a large number of fixed-blockages, which disrupt the cells from smooth spreading. Cells are often placed on top of the fixed-blockages in initial placement, and fixed blockages are density obstacles to prevent cells to pass over. If not properly handled, the wire length may grow dramatically by forcing cells moving out of blockages.

We use a contour-based density smoothing technique to alleviate the density obstacles as shown in Figure 2.4.4. First, we identify large blockages, which are those fixed macros with width and height larger than a certain threshold, such as 1% size of the chip size. In the beginning of the global placement, we adjust the density on bins covered by blockages, the adjusted density distribution is contour



(a) Initial adjusted densities for blockages (b) Blockage densities increase gradually during the placement

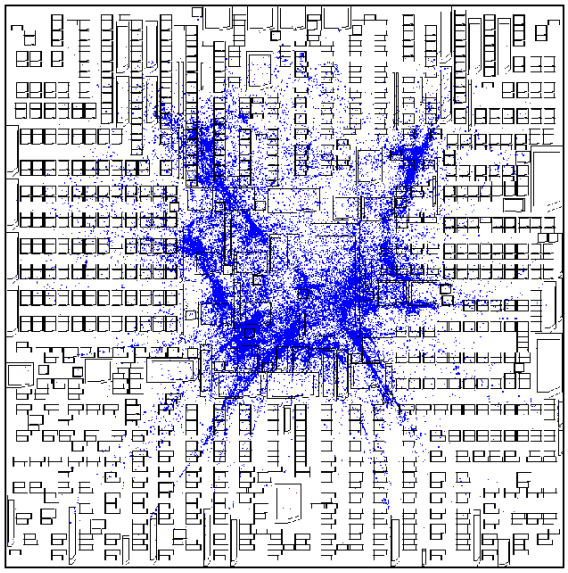
Figure 2.12: Dynamic density on blockages

based. For a bin covered by a big blockage, the bin density is set to be proportional to the distance of the bin to the blockage boundary. Therefore the highest density is in the bin lying in the middle of the blockage.

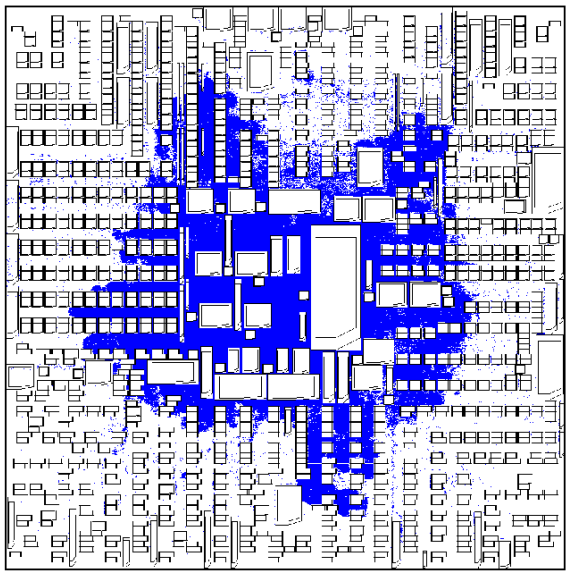
In the earlier stages of the global placement, the adjusted fixed blockage density is set to a very small value to allow cells to flow over. As the cells spreading stabilizes, the adjusted density increases gradually. The density in the middle of the fixed blockage rises to push overlapping cells out of blockages smoothly. The diffusion based pre-placement pushes cells over blockage easily according to the adjusted density distribution.

2.4.5 Wire length improvement heuristics

Beside the core techniques proposed above, there are more issues that will affect the final wire length quality. Pushing cells away from a region of congestion often contradicts with wire length optimization objective. Furthermore, Equation 2.2 optimizes the quadratic wire length, which is an indirect estimation of the linear wire length. The discrepancy between the quadratic approximation and HPWL is



(a) Initial placement. HPWL: 209×10^6



(b) After re-ordering. HPWL: 160×10^6

Figure 2.13: Improve the initial ordering of the placement

magnified in large-scale benchmarks, such as the ISPD2005 benchmarks, which contains a large amount of fixed macros. Figure 2.13(a) is the initial quadratic solution for circuit $adaptec4$. Without cell spreading, the unconstrained quadratic optimizer generates a solution of $2.09x10^7$ in HPWL, which is already worse than the final solution in Figure 2.13(b).

In quadratic placement, the wire length improving heuristics are crucial for the final HPWL results. The poor initial wire length implies that the initial ordering among cells are not ideal for HPWL, we interleave the medium improvement heuristic [86] and the anchor insertion based technique to generate a better initial cell ordering.

During the placement, wire length improving heuristics are employed between each iteration, which strongly affect the quality of the final HPWL. In DPlace, the quadratic optimization step is very fast and most of the CPU time is on wire length improving heuristics. In our experiments, the medium improvement heuristics used in FDP [86] was found effective in the earlier stages of the global placement. However, the medium improvement heuristic tends to create a lot of overlap in later iterations. The iterative local refinement technique [84] was found effective during the later spreading stages. At the point cells stop to spread, iterative local refinment can also be tuned to improve the density distributions. We use the iterative local refinement [84] to improve the density distribution and further reduce the wire length during the later stages of global placement.

2.5 Legalization and detailed placement

Legalization and detailed placement are non-trivial for the final wire length quality of the placer. Before legalization, we divide the placement region into regular bin structures and analyze the density overflow in each bin. We swap cells out of the overflowed bins and swap cells between bins if such a swap helps to further reducing the wire length. Once the bin density overflow is below a threshold, we run a Tetris [40] like legalization flow. We first legalize all movable macros such that no overlap exist between macros. Blockages/macros will split the placement region into row segments. We identify all row segments, sort cells and pack cells into the closet row segment with the minimum cost.

In this work, we use the FastDP [72] as the detailed placement engine to improve the wire length further.

2.6 Overall algorithm

The overall algorithm of our placer is summarized in **Algorithm** 1. In every global placement iteration, cells are diffused to reach a specified density distribution, and the anchor cell based wire length optimization is performed m times to reduce the wire length. The larger m, the shorter the wire length, and the worse the density distribution. Therefore, m is less than 3. We legalize the placement before using FastDP as the detailed placement for final wire length improvement.

Algorithm 1 The Overall Algorithm

The global placement
  Build matrix $\mathbf{A}$, and matrix $\mathbf{A}'$
  Generate an initial quadratic placement with matrix $\mathbf{A}$
  Improve the initial cell ordering $\mathbf{A}$
  Repeat
    Do diffusion based placement for k iterations
    Do m iterations
      Generate anchor cells and lock them at the gravity centers
      Compute HPWL net weights, update $\mathbf{A}'\mathbf{x} = \mathbf{b}$
      Solve $\mathbf{x} = \mathbf{A}'^{-1}\mathbf{b}$
    end
    if (In first a few iterations)
      Use medium improvement heuristic to repair wire length
    else if (Cells are roughly spread)
      Use iterative local refinement to repair wire length
  Until (reaches a desired density distribution)
  Further diffuse cells to remove remaining overlap
The legalization
  Legalize the macros, then legalize the standard cells
The detailed placement
  Use FastDP [72] as the detail placer

Table 2.1: Statistics on new Hessian $\mathbf{A}'$ and the Hessian $\mathbf{A}$ for conventional formulation, and the quadratic solver runtime comparisons

	Matrix $\mathbf{A}$				Matrix $\mathbf{A}'$				Solver
	Size	Entries	Precon	Solve	Size	Entries	Precon	Solve	speedup
adaptec1	243K	196K	15.85	4.65	211K	430K	0.53	0.19	24.5x
adaptec2	355K	2099K	25.61	7.38	254K	557K	0.90	0.30	24.6x
adaptec3	674K	3713K	38.18	15.61	494K	1131K	1.74	0.58	26.9x
adaptec4	508K	3676K	38.42	15.51	451K	997K	1.97	0.49	31.7x
bigblue1	392K	2287K	29.78	6.87	278K	603K	1.16	0.36	19.1x
bigblue2	729K	3937K	47.79	22.78	535K	1178K	2.29	0.82	27.8x
bigblue3	1389K	7290K	103.93	39.32	1096K	2714K	4.54	1.70	23.1x
bigblue4	2831K	16850K	221.47	75.70	2169K	5190K	10.66	3.91	19.4x
avg									24.6x

Table 2.2: Wire length and runtime comparison with FastPlace3.0, mPL6, Capo10.2, and APlace2.0 on ISPD2005 benchmark

	HPWL $\times 10^6$					Runtime(s)				
	D + F	FP3	mPL	Capo	APlace	**D + F**	FP3	mPL	Capo	Aplace
adapt1	78.534	1.011	0.991	1.162	1.001	606	0.69	5.13	10.46	14.99
adapt2	89.415	1.041	1.031	1.124	1.072	842	0.74	3.56	8.93	14.49
adapt3	221.817	0.982	0.962	1.031	0.982	1874	0.82	3.12	5.50	9.68
adapt4	198.506	1.014	0.974	1.045	1.055	1628	0.81	4.68	7.97	17.39
bigbl1	95.339	1.004	1.014	1.144	1.054	903	0.75	4.08	9.98	12.69
bigbl2	160.149	0.962	0.943	1.011	0.953	4656	0.43	2.93	4.99	7.52
bigbl3	363.073	1.046	0.952	1.099	1.130	5409	0.71	1.94	7.01	6.95
bigbl4	869.804	0.958	0.958	1.111	1.005	14026	0.41	1.72	5.61	6.85
Avg	**1**	1.002	0.978	1.091	1.031	**1**	0.67x	3.40x	7.56x	11.32x

Table 2.3: Wire length ($\times 10^6$) comparison with other placers in ISPD 2005 placement contest

Placers	adapt1	adapt2	adapt3	adapt4	bigbl1	bigbl2	bigbl3	bigbl4	ratio
DP + FD	78.53	89.41	221.82	198.51	95.34	161.15	363.07	869.80	**1.034**
Aplace	79.50	87.31	218.00	187.65	94.64	143.82	357.89	833.21	1.00
mFAR	-	91.53	-	190.84	97.70	168.70	379.95	876.28	1.06
dragon	-	94.72	-	200.88	102.39	159.71	380.45	903.96	1.08
mPL	-	97.11	-	200.94	98.31	173.22	369.66	904.19	1.09
FastPlace	-	107.86	-	204.48	101.56	169.89	458.49	889.87	1.16
Capo	-	99.71	-	211.25	108.21	172.30	382.83	1098.76	1.17
NTUP	-	100.31	-	206.45	106.54	190.66	411.81	1154.15	1.21
fs50	-	122.89	-	337.22	114.57	285.43	471.15	1040.05	1.50
K&D	-	157.65	-	352.01	149.44	322.22	656.19	1403.79	1.84

2.7 Experiments

We implement our placer in C++ and run the experiments on a Linux machine with 3.4 GHz 64-bit Xeon processors and 8G memory. We used the Hybrid solver [73] as our quadratic system solver, and the FastDP [72] as the detailed placer to further improve the wire length. Our current focus is to obtain a good wire length efficiently. We give the wire length and runtime results on ISPD 2005 benchmarks [66].

The anchor cell based formulation in DPlace gives significant advantage on the solving speed of the quadratic solver. Table 6.5 shows the statistics of the new Hessian matrix $\mathbf{A}'$ used in our placer, versus the Hessian matrix $\mathbf{A}$ in conventional formulation. Column *Size* shows the dimension of the Hessian, and column *Entries* shows the non-zero entries in the Hessian. Column *Precon*. shows the CPU time to preconditioning each Hessian matrix. Same preconditioning quality targets are used for the comparison. Column *Solve* shows the CPU time to solve one iteration of the quadratic system. Comparing with the conventional Hessian $\mathbf{A}$, the new Hessian $\mathbf{A}'$ is about 30% smaller on the dimension of the matrix. Furthermore, because $\mathbf{A}'$ is extremely sparse (Figure 2.10 and Table 6.5), the runtime to precondition and solve the new quadratic system are improved significantly. The quadratic solver achieved a 24x times speed up on solving time.

In Table 2.2, we compare DPlace with some of state of art academic placers, including the FastPlace3.0[85], mPL6 [18, 17], Capo10.2 [77] and APlace2.0 [53, 48] on the ISPD 2005 placement benchmark. Our placer and FastPlace3.0 are tested on the same machine and compared directly. The HPWL and runtime of mPL6,

Cap10.2 and APlace2.0 are derived from the FastPlace3.0 paper [85], in which they are directly compared with that of FastPlace3.0. Although not as accurate as running all placers on the same machine, we can still roughly compare the relative runtimes among all placers. For ISPD 2005 benchmark suite, the average HPWL result of DPlace (using the FastDP as the detailed placer) is 0.2% better than that of FastPlace3.0, and DPlace wins 5 out of 8 circuits. The average HPWL of DPlace is 2.2% higher than mPL6, 9.1% and 3.1% better than Capo10.2 and APlace2.0 respectively. The total runtime of DP+FD is about 33% longer than FastPlace3.0, 3.4 times faster than mPL6, 7.56 times faster than Capo10.2 and 11.32 times faster than APlace2.0.

We compare the HPWL results of DPlace with that of other placers in ISPD 2005 placement contest in Table 2.3. It is to be noted that the results in Table 2.3 were the best possible results generated by each placer, with no runtime limitation, for the ISPD 2005 placement contest. The ISPD2005 results of APlace1.0 are 3.4% better than our placer on average. However, the total runtime of APlace1.0 to finish all circuits are much longer, 113.2 hours for 6 circuits, on a 1.6GHz computer [48], compared with 8.3 hours for 8 circuits in our case, on a 3.4GHz computer. Other than Aplace, DPlace generates the best results among all other placers for their ISPD2005 placement contest versions.

2.8 Summary

In this chapter, we present DPlace, a new analytical placement tool for large scale placement. DPlace is based on the diffusion placement technique to spread

cells smoothly, which generates a golden placement for improved density distribution. Then DPlace uses the anchor cells based formulation as well as wire length improvement heuristics to reduce the wire length.

In the wire length reduction stage of DPlace, the Hessian matrix of the anchor cells based quadratic formulation is extremely sparse. In our framework, since it is possible to affix explicit cell movement control in the diffusion stage, our new formulation has the potential advantages for ECO and timing driven placement, in which precise cell movement control is required. By using the FastDP as the detailed placement, the HPWL results DPlace are the best among published quadratic placement works, and close to the best reported results on ISPD 2005 placement benchmark suite. Also, the runtime of DPlace is much better than most of state-of-art placers.

Chapter 3

Computational Geometry Based Placement Migration

3.1 Introduction

The nature of a physical synthesis flow is highly iterative, and the global placement is never a complete solution for design closure. In modern placement and physical synthesis of VLSI circuits, one is increasingly faced with the placement migration problem, which is to take an existing placement, fix some design violations and re-legalize it. For example, during physical synthesis or Engineering Change Order (ECO) optimization, many buffers may be inserted and gates resized, creating a lot of overlapping cells. These cells need to be legalized, but one should avoid disturbing the previous placement too much to achieve design convergence. Also another example, post routing congestion analysis may identify severe hot spots (e.g., congestion, noise, power, thermal), and placement migration is needed to smoothly spread out cells in these hot spots [74]. Due to the complexity of modern nanometer designs, it is unlikely to design one placement algorithm that meets the multi-objective design closure target in a single run. More often, a placement flow involves multiple placement-improvement iterations. So a stable placement migration algorithm is crucial for the multi-objective design closure.

These tasks share a common theme of starting with an initial placement that

is "good" and perturbing it so that it is improved in some way while still preserving the essential characteristics (cell ordering, wirelength, etc.) of the original placement. Ideally, the later placement iteration should be able to preserve previous fixes and accumulate additional improvements to achieve the design closure. Therefore, the stability of the placement algorithm is very important. Obviously, we do not want each placement iteration generates entirely different result and destroys all previous optimization efforts.

Among various placement migration applications, legalization is probably the most common one. Therefore, the remainder of the paper will discuss our placement migration algorithm in this context. Existing legalization techniques for legalization include network flow [57, 13], dynamic programming [1, 52], heuristic ripple cell movement [42], and single row optimization [49, 14]. The network flow approach [13] uses minimum cost flows to minimize the weighted sum of (squared) cell movements. The dynamic programming based approach [1] solves the optimal assignment of cells to placement sites under the constraint of cell ordering. Mongrel [42] uses a greedy heuristic to move cells from overflowed bins to under capacity bins in a ripple fashion based on total wire length (TWL) gain. The single row optimization techniques [49, 14] use dynamic programming to optimally place cells in a single circuit row.

While there are many existing legalization algorithms, there are very few works directly targeting incremental and stable placement migration.In this chapter, we develop a novel technique for stable placement migration based on the *computational geometry*. We also propose a new placement stability *metric* which can

be used to measure the placement migration stability. Our algorithm has two key steps: bin-based cell spreading and Delaunay triangulation based overlap reduction. The algorithm takes advantage of the computational geometry property of the existing placement. Thus it captures the relative cell order nicely during placement migration. Our experimental results compared to other widely used legalization algorithms clearly demonstrate the superiority of our algorithm, with over 10% wire length reduction and significantly better stability score.

The rest of the chapter is organized as follows. Section 3.2 presents the bin based spreading algorithm. Section 3.3 presents the Delaunay based overlap reduction procedure. The complete computational geometry based legalization algorithm is given in section 3.4. Section 3.5 proposes a new placement stability metric suitable for placement migration. Very promising experimental results are obtained in section 3.6, followed by summary in section 3.7.

3.2 Bin Based Spreading

A placement is close to legal if all that is required to legalize the placement is to snap cells to rows or perhaps perform minor cell sliding in order to fit the cells. Assuming the chip layout is divided into equal sized bins, the placement is considered close to legal if the area density of every bin is less than or equal to D_{max} (e.g., $D_{max} = 1$). For all bins with density greater than D_{max}, cells must be migrated to other bins. Therefore the goal of our migration algorithm is to reduce the density of each bin to no more than D_{max} while avoiding moving these cells far from their original locations thus preserving the original placement characteristics.

Bin based spreading is a geometric approach to evenly reduce cell density on the congested regions. Suppose we divide the entire placement region into K*L square bins, there will be $(K+1)$*$(L+1)$ bin corners. The idea is to move those bin corners such that the resulting bin capacity would satisfy the density constraints, and then move cells accordingly. By stretching the bin corners, we preserve the relative order of neighboring bins; meanwhile by interpolating cells relative to its bin corners, we preserve the relative order of cells inside the bin. We perform the bin stretching and cell interpolation iteratively until all the bins are under the maximum density D_{max}.

3.2.1 Bin Stretching

At each iteration, we first compute the bin density $D_{k,l}(n)$ (the nth iteration), then compute the amount of stretching needed for each bin. For those overpopulated bins, the idea is to expand that bin such that the density of the new bin is equal to D_{max}. At the same time, to accelerate the spreading process, we allow the adjacent bins to shrink such that their densities equal to D_{max} as well. The amount of stretching for bin (k,l) on both horizontal and vertical directions can be written as:

$$\begin{aligned} \varepsilon_{k,l}^{x} &= (\sqrt{\frac{D_{k,l}(n)}{D_{max}}} - 1)W \\ \varepsilon_{k,l}^{y} &= (\sqrt{\frac{D_{k,l}(n)}{D_{max}}} - 1)H \end{aligned} \tag{3.1}$$

where W and H are the bin width and height, respectively.

Stretching each bin itself would generate overlaps between adjacent bins.

Therefore we stretch the bin corners of adjacent bins instead of bins itself. Let $(p^x_{k,l}(n), p^y_{k,l}(n))$ denotes the coordinates of an inner bin corner, which is shared by four neighboring bins, denoted as $(k-1,l-1)$, $(k-1,l)$,$(k,l-1)$, and (k,l). We can use (3.1) to compute the amount of horizontal and vertical stretching needed for each one of the four bins, which will give us four stretched corner positions, and then compute the combined number of these four as the corner position for next iteration, $(p^x_{k,l}(n+1), p^y_{k,l}(n+1))$,

$$
\begin{aligned}
p^x_{k,l}(n+1) &= p^x_{k,l}(n) + 0.5(\varepsilon^x_{k-1,l-1} + \varepsilon^x_{k-1,l} - \varepsilon^x_{k,l-1} - \varepsilon^x_{k,l}) \\
p^y_{k,l}(n+1) &= p^y_{k,l}(n) + 0.5(\varepsilon^y_{k-1,l-1} + \varepsilon^y_{k,l-1} - \varepsilon^y_{k-1,l} - \varepsilon^y_{k,l})
\end{aligned} \tag{3.2}
$$

Because the stretching is uniform on both bin corners on the same bin edge, we only take half the stretching value given by (3.1). If any neighboring bin is on the chip boundary, we take the 0.5 factor off.

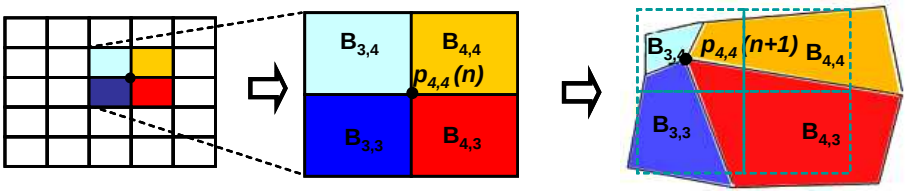


Figure 3.1: Illustration of bin and corner stretching

Figure 3.1 is an illustration of the movement of the corner point $p_{4,4}$ under accumulated stretching from all four surrounding bins. Bin $(3,3)$, $(4,3)$, and $(4,4)$ are over the maximum density, therefore we expand them, while bin $(3,4)$ is under the maximum density, thus we compact it. We will have four new corner positions of this corner for each bin. Such process is iterated as needed. After computing

coordinates of all points, cells inside the bin will move within the distorted bin as explained in next section.

3.2.2 Cell Interpolation

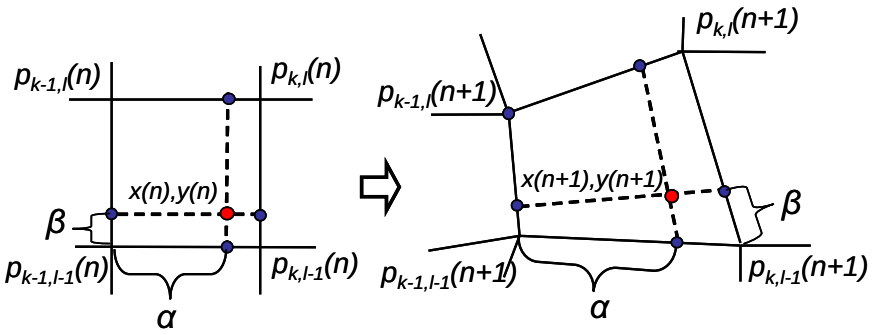


Figure 3.2: Cell location interpolation on stretched bin

The computation of new cell coordinates is a linear interpolation process, which maps all cells from the original bin into the new bin at the same relative positions. As shown in Figure 3.2, the four corner coordinates of the bin are $\mathbf{p}_{k-1,l-1}(n)$, $\mathbf{p}_{k,l-1}(n)$, $\mathbf{p}_{k-1,l}(n)$, and $\mathbf{p}_{k,l}(n)$. Their coordinates after bin stretching are: $\mathbf{p}_{k-1,l-1}(n+1)$, $\mathbf{p}_{k,l-1}(n+1)$, $\mathbf{p}_{k-1,l}(n+1)$, and $\mathbf{p}_{k,l}(n+1)$. For a cell $(x(n),y(n))$ within the bin, the new coordinates $x(n+1)$ and $y(n+1)$ can be computed by the following equations.

$$\begin{aligned} x(n+1) &= \gamma_x + \beta(\xi_x - \gamma_x) \\ y(n+1) &= \gamma_y + \alpha(\xi_y - \gamma_y) \end{aligned} \tag{3.3}$$

where

$$
\begin{aligned}
\alpha &= \frac{x(n) - p^x_{k-1,l-1}(n)}{p^x_{k,l-1}(n) - p^x_{k-1,l-1}(n)} \\
\beta &= \frac{y(n) - p^y_{k-1,l-1}(n)}{p^y_{k-1,l}(n) - p^y_{k-1,l-1}(n)} \\
\gamma_x &= p^x_{k-1,l-1}(n+1) + \alpha(p^x_{k,l-1}(n+1) - p^x_{k-1,l-1}(n+1)) \\
\xi_x &= p^x_{k-1,l}(n+1) + \alpha(p^x_{k,l}(n+1) - p^x_{k-1,l}(n+1)) \\
\gamma_y &= p^y_{k-1,l-1}(n+1) + \beta(p^y_{k-1,l}(n+1) - p^y_{k-1,l-1}(n+1)) \\
\xi_y &= p^y_{k,l-1}(n+1) + \beta(p^y_{k,l}(n+1) - p^y_{k,l-1}(n+1))
\end{aligned}
\tag{3.4}
$$

3.2.3 Bin Based Spreading Algorithm

At each iteration of the bin based spreading algorithm, it first stretches the bin corner to make congested bin larger, then interpolates cell locations accordingly. It then restores all the bin boundary and starts an new iteration. The new iteration recomputes the bin density and repeats all above procedures. The process stops once that all the bin densities are lower than the maximum density D_{max}.

To avoid over expansion in non-congested region, we only change the bin corners of those bins above D_{max} during bin stretching. It assures that cells are pushed from high density area to low density area steadily and smoothly. It also reduces unnecessary oscillation and computations.

The stability of the migration process is affected by the bin size (area) as well. The ideal initial bin size is depending on the size of the circuits. If the bin

size is too large, the internal density distribution inside the bin might still violate the density constraints even if the bin as a whole is under D_{max}. However, if the bin size is too small, oscillation will appear and bin boundary distortion may impact the smoothness of spreading. We may see cells tend to cluster in some areas. This problem is solved by a hierarchical addition to our original formulation. The idea is straightforward. It uses big bin sizes from at the beginning, then recursively cuts big bins into smaller bins, and adjusts the internal density distribution. The hierarchical technique is necessary to handle fixed macros. At the time the bin size is smaller enough, bin edges be close to macro boundaries. Cells will move along the boundary, they will not move toward the macros. The complete bin based spreading algorithm is given by Algorithm 2.

Note that our approach is different from the grid warping [88] and cell shifting [84]. At each partition step, grid warping slices the region into 2 x 2 or 4 x 4 equal "volume" quadrilateral grids, transforming the grid (and cells) back to equal shape rectangles to form the subproblems. The elastic grids in grid warping are the equivalence to Gordian's min-cut partition [55], both purpose is for partition, while our bins are used for spreading directly. We reshape each bin individually at each step and rely on iterations to flow cells out eventually. The cell shifting [84] technique is an one dimensional greedy shifting, which is used to generate the spreading forces for the global placer. It is the quadratic solver that does the actual spreading; while our approach is a two dimensional approach, and it spreads out cells directly.

Algorithm 2 Computational Geometry Bin Based Spreading

Procedure: BIN
Input: cell placement x_i, y_i, bin area $A_B = W \cdot H$, maximum bin density D_{max}
Output: new placement $\hat{x}_i, \hat{y}_i$
begin
  Initialize bin density $D_{k,l}$;
  if A_B is too small **then return;**
  while any $D_{k,l} > D_{max}$
    for each bin with $D_{k,l} > D_{max}$
      Compute bin expansion $\varepsilon^x_{k,l}$, $\varepsilon^y_{k,l}$ with (3.1);
    end for
    Compute bin corner $\mathbf{p}_{k,l}(n+1)$ with (3.2);
    Interpolate cell locations $x_i(n+1), y_i(n+1)$ with (3.3);
    Restore all bin corners, update $D_{k,l}$;
    $n = n+1$;
  end while
  Update $\hat{x}_i = x_i(n), \hat{y}_i = y_i(n)$;
  Reduce bin area $A_B = A_B/2$;
  Recursively call BIN($\hat{x}_i$, $\hat{y}_i$, A_B, D_{max});
end

3.3 Delaunay Triangle Based Overlapping Removing

Bin based spreading is good for coarse level spreading. However, to further remove overlapping between cells, we need to use more fine-grained migration techniques. In this section, we will present the Delaunay triangulation based algorithm to effectively remove cell overlap while preserving placement stability.

3.3.1 Delaunay Triangulation

The Delaunay triangulation is the dual of the Voronoi diagram – one of the most fundamental data structures in computational geometry [36]. The Voronoi diagram for a collection of geometric objects is a partition of space such that each

of them consists of the points closer to one particular object than to any others. It contains a straight-line edge connecting two sites in the plane if and only if their Voronoi regions share a common edge. The Delaunay triangle edges of an object essentially captures its relative proximity relationship with other objects.

Figure 3.3 shows an example of the Voronoi diagram and its corresponding Delaunay triangulation. For a given VLSI placement to be migrated smoothly to another solution due to legalization need, congestion or noise mitigation, we can compute the Delaunay triangulation for all cells efficiently. Based on this Delaunay triangulation that captures the "preferred" proximity relationships among all fixed and placeable objects, we can perform stable placement migration, to spread cells smoothly from congested area, as illustrated in Figure 3.3.

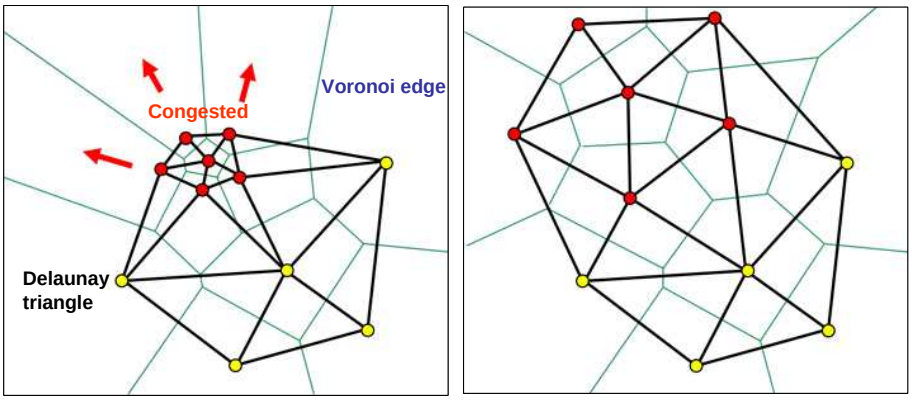


Figure 3.3: Delaunay triangulation captures the relative order, which can be used to spread cells during placement.

Delaunay triangulation is an important topic in computational geometry and has wide applications in varies field, such as visualization, finite element analysis, and discrete wireless networks. There are quite a few mature Delaunay triangulation

algorithms developed, with the computational complexity ranges from O(nlog$n$) to O(n^2). The reader is referred to [36] for a comprehensive survey of Delaunay triangulation and Voronoi diagram.

Given a placement, we can construct the Delaunay triangulation of all the cells using its center locations as triangle nodes. Then the placement plane becomes a planar graph $G = (V,E)$, with $V = \{v_1, v_2, ..., v_n\}$ corresponding cells and $E = \{e_1, e_2, ...e_m\}$ triangle edges. The boundary of the graph are fixed pads. We only move non-boundary or non-fixed cells. Figureo 3.4 shows a Delaunay triangulated placement region.

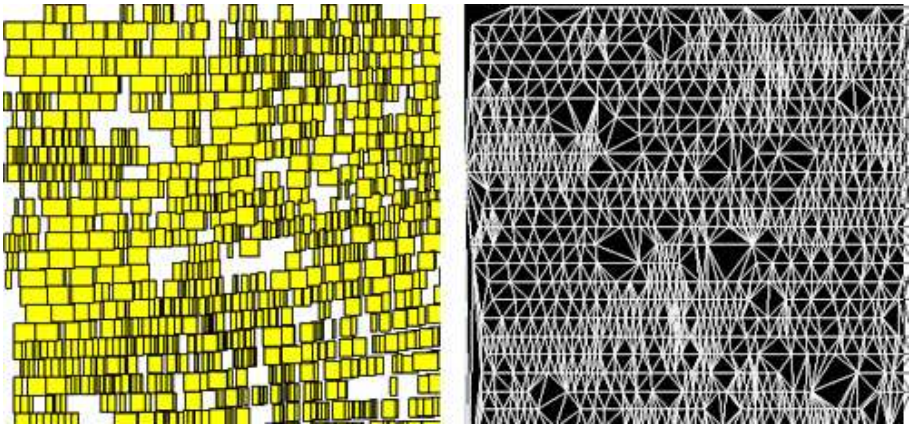


Figure 3.4: Delaunay triangulation of a placement region

3.3.2 Fine-grain Overlapping Reduction

Because Delaunay triangulation helps to identify all close neighbors of one cell, such detailed information is valuable for fine-grain adjustments. We use the Delaunay triangulation to do further cell spreading, where the bin based spreading is

not applicable. The Delaunay triangulation based cell overlapping reduction works as follows.

To iterate through cells in the placement order, we build a tree structure on the delaunay triangulated placement. One cell in the center of the placement is selected as the tree root, and all cells connecting to the root by Delaunay edges are added into the tree as the second level tree nodes. Then all cells connecting to second level nodes are added as the third level tree nodes. Note that one cell may connect to two second level tree nodes by Delaunay edges. The cell is added to one tree node as the child only. The criteria of where to add the cell is to keep the number of child of each tree node balanced. Similarly, the tree keeps growing until all cells in the placement are added. Figure 3.5 illustrats the steps to build the tree on a delaunay triangulated region. Cells with the same color are tree node the same level.

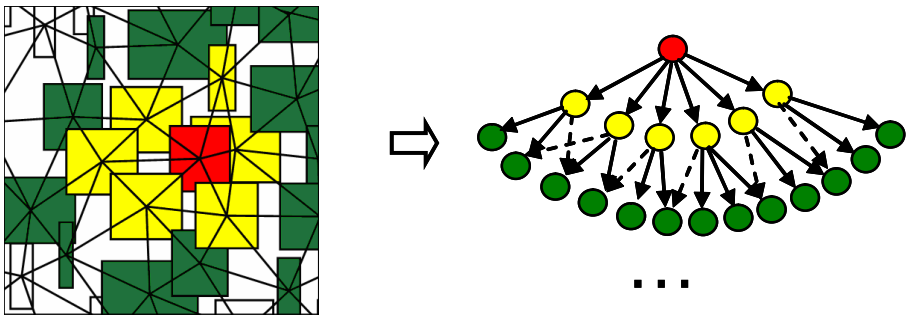


Figure 3.5: Tree structure for Delaunay edge traversing

Starts from the root, the algorithm traverses the tree in breadth-first manner. For every tree node - cell i, all Delaunay edges connecting cell i with the same

or next level nodes are inspected. Let $e_{i,j}$ be the Delaunay edge between cell i and cell j. From the Delaunay triangle properties, we know that i and j are the nearest neighbor to each other. If cell i does not overlap with cell j, we do nothing and move on to the next Delaunay triangle edge. If cell i overlaps with cell j, the overlap distances on x and y directions are measured and cells will be pushed away accordingly. Let $\triangle_{i,j}^{x}$ and $\triangle_{i,j}^{y}$ be the x and y direction overlapping between i and j, respectively. If $\triangle_{i,j}^{x} > \triangle_{i,j}^{y}$, a repelling force is generated between cell i and cell j on x direction. We try to make minimum movement to remove the overlapping. So the force is inversely proportional to the cell sizes with weight to push the cell away from congestion. Let $f_{i->j}^{x}$ denote the repelling force from cell i to cell j.

$$f_{i->j}^{x} = \triangle_{i,j}^{x} \frac{w_j}{w_i + w_j} \tag{3.5}$$

where w_i and w_j are the widths of cells i and j. If $\triangle_{i,j}^{x} < \triangle_{i,j}^{y}$, the fource will be in the y-direction, i.e.,

$$f_{i->j}^{y} = \triangle_{i,j}^{x} \frac{h_j}{h_i + h_j} \tag{3.6}$$

where h_i and h_j are the heights of cells i and j.

If a movable cell i is connected with multiple neighbors by Delaunay edges, the total force F_i^x on $cell_i$ is the superposition of all overlapped neighboring cells

$$F_i^x = \sum_{j \in Neighbor(i)} f_{j->i}^{x} \tag{3.7}$$

where $Neighbor(i)$ denotes the set of cells overlapped with cell i.

Figure 3.6 is an example to illustrate how forces are added to the overlapping cells. As shown in Figure 3.6, assume cell A, B, C are within one Delaunay triangle. We can see that B and C are overlapped, and the overlapping in x direction is smaller, i.e. $\triangle^x_{B,C} < \triangle^y_{B,C}$. Then the y-directional force $f^y_{B->C}$ and $f^y_{C->B}$ will be applied on cells C and B, respectively. In the case that a cell overlaps with many surrounding neighbors, the total force tends to cancel each other. This usually happens at the center of congested area, and we can set certain density threshold to avoid redundant computation. The cells close to whitespace will move first and pull cells inside congested areas out smoothly.

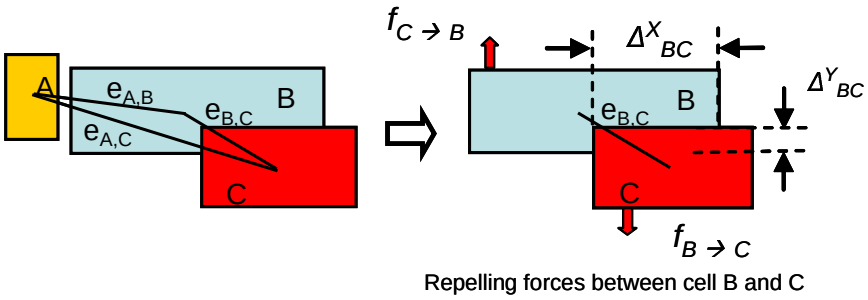


Figure 3.6: Delaunay force to reduce overlapping

The Delaunay triangulation based overlapping reduction process is outlined in Algorithm 3.

3.4 Computational Geometry based Legalization

Our algorithm consists of two major steps: bin based spreading and Delauney Triangle based overlap reduction. As described earlier, bin based spreading

Algorithm 3 Delaunay Based Overlapping Reduction

Procedure: DELT
Input: cell placement x_i, y_i
Output: new placement $\hat{x}_i, \hat{y}_i$
begin
  while stopping criteria is not satisfied
    if redo Delaunay condition is satisfied **then**
      $T = \{V, E\} \leftarrow (x_i, y_i)$
    end if
    BFS (T)
      for each edges $e_{i,j}$ connect with cell$i$
        check connected cells i and j
        if i does not overlap with j **then** continue;
        if $\triangle_{i,j}^{x} < \triangle_{i,j}^{y}$
          compute $f_{i \to j}^{x}$
        else
          compute $f_{i \to j}^{y}$
        end if
      for each cell i in T
       move all cells on force
    end BFS
    sum up forces and update coordinates of cell
    end for
  end while
end

reduces the bin density overflow quickly at coarse level, and the Delaunay Triangle based overlap reduction step works at fine-gained level to reduce the overlap between adjacent cells. After bin based spreading and Delaunay Triangle based overlap reduction, the placement should have a max density of D_{max} and is roughly legal. We will run a final legalization step to put cells onto circuit rows without overlap,which takes very small effort since the density constraint is satisfied at fine granularity level. The emphasis is to study the impact of our computational ge-

ometry placement migration algorithms comparing with other methods, such as the greedy and the flow, thus we just use a standard legalizer to generate the final legal placement. In fact, it is almost trivial after our migration. The complete computational geometry based legalization (*CGL*) algorithm is given in Algorithm 4). Note that the combined bin based spreading and Delauney Triangle algorithm gives the best result. For comparison purpose, we test the bin based spreading algorithm alone for legalization. It is referred to as CGL_B.

Algorithm 4 Computational Geometry Legalization Algorithm

Procedure: CGL
Input: A cell placement x_i, y_i
Output: A new placement $\hat{x}_i, \hat{y}_i$
Parameters: Initial bin area: A_B, max bin density D_{max}
begin
  Call bin based spreading algorithm (Algorithm 2):
    ($\hat{x}_i$, $\hat{y}_i$)= BIN(x_i, y_i, A_B, D_{max});
  Call Delauney Triangle based algorithm (Algorithm 3):
    ($\hat{x}_i$, $\hat{y}_i$)=DELT($\hat{x}_i$, $\hat{y}_i$);
  Put cell onto circuit row and remove remaining overlaps;
  return $\hat{x}_i$, $\hat{y}_i$
end

3.5 Geometric Placement Stability Metrics

During placement migration one often needs to compare the difference of the original (golden) placement with a new placement generated by placement migration. It can be measured by the placement stability metrics. In the existing literature [8], two placement stability metrics are used: one measures the average cell movement between two placements, and the other measures the change of net

clusters. During placement migration, however, it is possible that a large number of cells are shifted, all with a small amount. Thus all nets between cells have very small changes (like in our Delaunay triangulation spreading). For such scenario, the *absolute* cell movement metric is not a good metric [8]. The net cluster metric [8] is good to capture global placement stability where one big cluster can be moved to another part of the chip. But it is not very suitable for placement migration applications where most changes are small. During placement migration, it is desired to keep the relative geometric order and punish the most disruptive changes. Therefore we propose the following geometric stability metric.

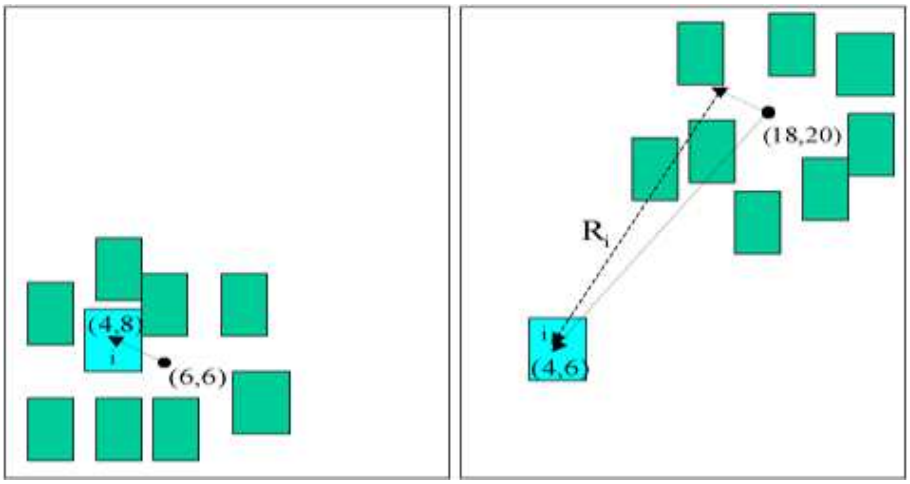


Figure 3.7: Relative distance of cell *i*.

Suppose we are given two placements: a golden placement *A* and a new placement *B* generated by placement migration. Our idea is to measure the change of cell placement *relative* to its neighboring cells and sum up the most significant changes to capture the difference of these two placements. Both placements have

the same number of cells. The coordinates of cell i are (x_i, y_i) and $(\hat{x}_i, \hat{y}_i)$ for placement A and B, respectively. We select a group of cells adjacent to cell i in placement A, and compute the **g**eometric **c**enters (GC) of this group in both placement A and B as (x_i^{GC}, y_i^{GC}) and $(\hat{x}_i^{GC}, \hat{y}_i^{GC})$ as follows.

$$\begin{aligned} x_i^{GC} &= \frac{max(x_j) + min(x_j)}{2} \\ y_i^{GC} &= \frac{max(y_j) + min(y_j)}{2} \\ \hat{x}_i^{GC} &= \frac{max(\hat{x}_j) + min(\hat{x}_j)}{2} \\ \hat{y}_i^{GC} &= \frac{max(\hat{y}_j) + min(\hat{y}_j)}{2} \end{aligned} \tag{3.8}$$

where j refers to cells within a certain Euclidian distance to cell i in placement A. We can then define the *relative* movement of a cell i from placement A to B as the squared Euclidian distance of the relative positions of cell i to $(x_i^{GC}, y_i^{GC})$and $(\hat{x}_i^{GC}, \hat{y}_i^{GC})$, as shown in following equation:

$$R_i = [(\hat{x}_i - \hat{x}_i^{GC}) - (x_i - x_i^{GC}]^2 + [(\hat{y}_i - \hat{y}_i^{GC}) - (y_i - y_i^{GC})]^2 \tag{3.9}$$

The new placement stability metric R_i for each cell i essentially captures the relative change w.r.t. to its geometric neighborhood. Figure 3.7 shows two placements of cell i and its adjacent cells in the left placement, assuming the left placement is the original placement and the right one is after placement migration. Suppose originally cell i is placed at $(4,8)$ and the geometric center of its adjacent cells is at $(6,6)$. After placement migration, cell i is moved to $(4,6)$ while all of its original neighbors are shifted to upper right corner with a center at $(18,20)$. The relative positions of cell i to the center of its adjacent cells are shown as vectors

in Figure 3.7. Although the absolute location of cell i does not change much, the relative distance actually changes a lot, $R_i = [(4-18)-(4-6)]^2 + [(6-20)-(8-6)]^2 = 400$ as given by Eqn. (3.9), which can not be captured by the absolute cell movement between both placements. Note that we use squared distance to emphasize the impact of larger moves, since wiring delay is a quadratic function of wirelength. Larger moves will have higher possibility to degrade the overall timing closure.

Naively one may sum up R_i for all cells to measure the total geometric stability. However, it is the most disruptive changes of the relative order that have the biggest impact on the placement migration quality. Therefore, we set some filter and only count the top percent of cells in terms of R_i, e.g., top 1% cells. So, the overall geometry stability metric S_G can be written as

$$S_G = \frac{\sum_{j \in \mathbf{C}_{1\%}} R_j}{N} \tag{3.10}$$

where $\mathbf{C}_{1\%}$ is the set of top 1% cells with the largest R_j values and N is the number of cells in $\mathbf{C}_{1\%}$. Bigger S_G value means placement B is less similar to the original placement A, and more cells are placed away from their original affinity logics, thus more vulnerable to performance degradation.

3.6 Experimental Results

We implemented the computational geometry placement migration algorithms in C++ on a 3.4 GHZ Xeon Linux box, and use the modified *ISPD*02 benchmark [44] to test it. All the testcases of this benchmark place cells with equal den-

sity (95%) over the entire chip. To make it similar to the real industry placement after physical synthesis, we first linearly scale down each cell such that the entire chip density for each testcase is reduced to 80%. We then run a detailed placement algorithm to reduce the wirelength. After that, the placement is no longer equally distributed. To test the legalization algorithm, we generate the overlaps by expanding cells in the center of the chip. We linearly expand 15% of cells in the center by 67%. So after expansion, the overall chip density is increased to 90% and this original placement is no longer legal.

We compare the computational geometry based legalizer (*CGL* and CGL_B) to an industry greedy legalizer (*GRDY*) which uses slide-and-spiral techniques to place cells onto their nearest legal locations and to an industry network flow legalizer (FLW) which uses min-cost flow algorithm to direct cell movements. FLW is an industrial strength legalizer similar to [13]: first, cells are roughly spread out by the min-cost flow algorithm; then, they are moved to their final positions such that all overlaps are removed. *GRDY* sorts all the cells and place them sequentially. It first tries to place a cell at the original location. If that location is occupied, it performs a spiral search starting from the original location. During a spiral search, it could slide other placed cells a little bit in order to fit in.

The *CGS* and CGS_B experiments in Table 3.1 were tested on a 3.4 GHZ Xeon Linux box. Table 3.1 reports the TWL, stability and CPU time results of these legalizers. The TWL numbers are scaled to the TWL of the original illegal placement. Both CGS_B and *CGS* get much better TWL than *GRDY* and FLW. The improvement is over 10% on average. Both CGS_B and *CGS* get order of magnitude

better scores than *FLW* and *GRDY*. And *CGS* can further reduce it by 16% than CGS_B due to the Delaunay triangle based overlap reduction. Also we can see *CGS* is much faster than *GRDY* and *FLW* (over 10x speedup for bigger circuits).

Table 3.1: The wirelength, stability, and CPU time comparison with computation geometry/Delaunay based migration, followed by the legalization engine in the IBM environment (see details in [60]).

	TWL Comparison				S_G Comparison				CPU(s)		
	GRDY	FLW	CGS_B	CGS	GRDY	FLW	CGS_B	CGS	GRDY	FLW	CGS
ibm01	1.282	1.314	1.192	1.191	52199	37785	12685	11349	11	10	10
ibm02	1.056	1.064	1.013	1.013	44735	24126	5225	5172	23	25	23
ibm03	1.101	1.096	1.058	1.061	58375	52356	41099	26081	28	26	25
ibm04	1.165	1.195	1.096	1.100	119647	58967	52199	39039	62	29	30
ibm05	1.016	1.018	1.014	1.014	55040	65344	53387	49538	33	37	40
ibm06	1.111	1.123	1.036	1.035	60242	48890	4555	4750	69	53	35
ibm07	1.141	1.139	1.040	1.039	131024	119809	5232	5764	162	150	55
ibm08	1.154	1.153	1.043	1.042	153395	119758	3610	3750	307	242	65
ibm09	1.211	1.221	1.071	1.069	168791	154334	20743	16126	325	289	75
ibm10	1.180	1.181	1.032	1.030	302222	238185	5057	4740	806	575	89
ibm11	1.193	1.187	1.054	1.056	224925	190321	32519	50898	617	490	95
ibm12	1.162	1.167	1.051	1.050	361702	331289	5270	5088	1299	807	121
ibm13	1.229	1.233	1.064	1.059	370440	349016	177377	117833	939	734	134
ibm14	1.239	1.235	1.049	1.047	557898	456627	4581	4364	3654	2240	231
ibm15	1.274	1.273	1.067	1.065	698634	746372	130784	123811	5210	3643	430
ibm16	1.313	1.318	1.041	1.040	937453	746372	4435	4129	9927	6280	511
ibm17	1.267	1.285	1.043	1.040	1169281	1087672	14648	11019	11363	8280	558
ibm18	1.296	1.318	1.055	1.054	1116867	985042	3224	3267	12724	9160	661
Avg	1.188	1.197	1.057	1.056	365715	334406	31855	26858	2642	1837	177

To further understand the difference of geometric stability result of these approaches, Figure 3.8 and 3.9 show the relative distance histogram after legalization on *ibm*01. Figure 3.9 is a zoom-in view of the top 1% cells in Figure 3.8 to make it difference clear. *CGL* has less larger relative distances than *FLW* and *GRDY*, which means *CGL* would keep cells closer to their original neighbors, preserving the relative order. On the other hand, *GRDY* and *FLW* have less number of smaller relative distances than *CGL*, which means *CGL* tends to move more cells a smaller distance to avoid bigger moves. For placement migration applications, one often wants to limit the bigger moves but not care too much about the smaller moves,

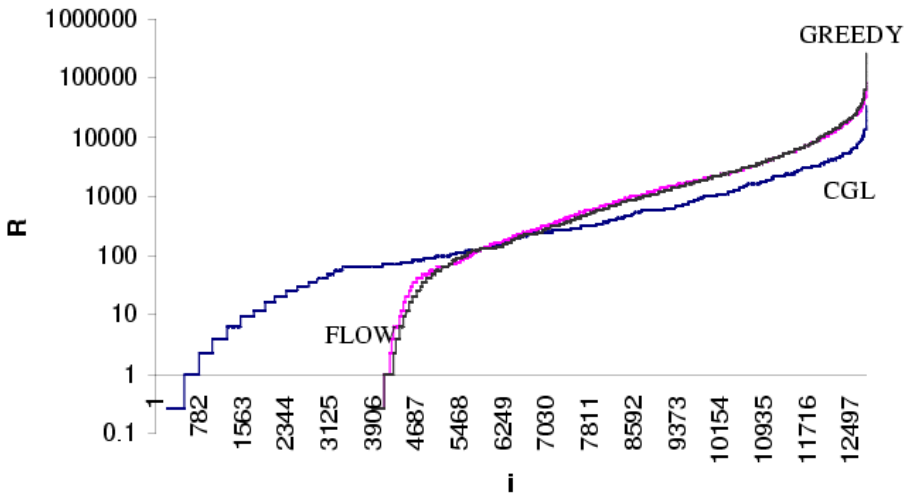


Figure 3.8: Histogram of R_i from three legalizers on *ibm*01.

therefore *CGL* is well targeted for those applications. The runtime comparison of is also reported in Table 3.1. We can see *CGL* is much faster than *GRDY* and *FLW* (over 10x speedup for bigger circuits). Therefore, it is both effective and fast.

In Table 3.2, we show the complete migration and legalization results from our tool. We use FastPlace1.0 [84] to generate an initial legal placement, then replace the cell size file with the 15% inflated version, as described earlier, to generate overlaps. We compared the Delaunay based legalizer CGL with two publicly available legalizers, *FPDP*1.0, the FastPlace 1.0 detailed placer [84], and *FSDP*5.0, the Fengshui 5.1 detailed placer [3]. Table 3.2 reports the wirelength, stability and CPU time comparison. The second set of experiments were run on on a quad-core 64-bit 3.4 GHz Xeon Linux machine. From Table 3.2, we see legalizer performs with or without considering placement stability.

We also implemented the diffusion-based legalization algorithm [75] for

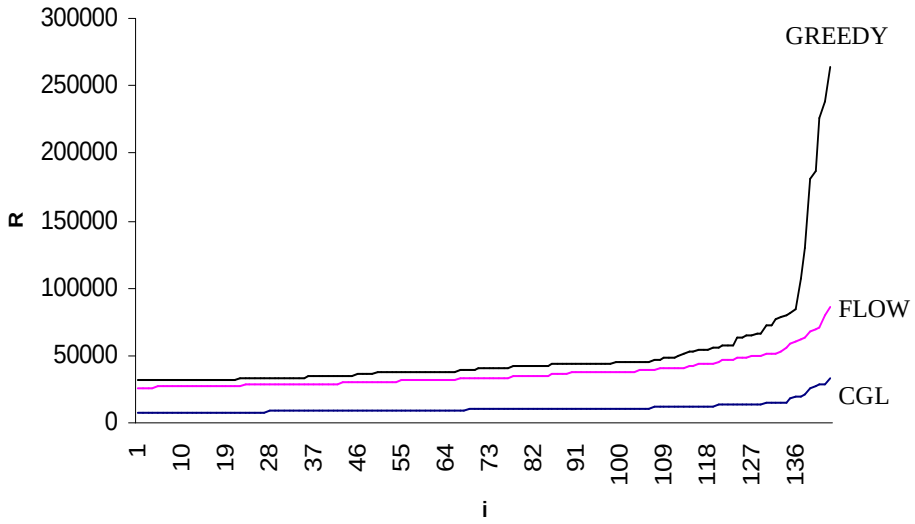


Figure 3.9: Histogram of top 1% R_i from three legalizers on *ibm*01.

the academic benchmarks. Our initial experience is that the diffusion algorithm produces slightly better result while computational geometry based algorithm is faster. However, it is not straightforward to make a fair comparison between them at this stage, because the results depend on the tuning. In general both algorithms share a common smooth spreading nature and generate comparable results.

3.7 Summary

The incremental nature of design optimization demands smooth and stable placement mitigation techniques. They must be capable of spreading cells to satisfy design constrains such as image space, routing congestion, signal integrity and heat distribution, while keeping the original relative order. To address these challenging tasks, we propose a novel computational geometry based placement migration framework. Our experimental results on legalization problem have demonstrated

Table 3.2: Wirelength, stability and CPU comparison of our Delaunay-based migration/legalization tool and the two publicly available detailed placement engines from FastPlace and Fengshui.

	TWL Comparison			S_G Comparison			*CPU* (s)		
	FPDP1.0	FSDP 5.1	CGDP	FPDP1.0	FSDP 5.1	CGDP	FPDP1.0	FSDP 5.1	CGDP
ibm01	1.182	1.074	1.117	35523	3116	1147	2	12	3
ibm02	1.069	1.053	1.086	29612	4169	1727	2	19	4
ibm03	1.356	1.051	1.100	131172	10626	2529	5	20	5
ibm04	1.457	1.120	1.137	130771	73378	35760	5	26	6
ibm05	1.153	1.131	1.098	151385	72075	9623	6	36	6
ibm06	1.236	1.071	1.116	59274	5213	2061	5	39	7
ibm07	1.182	1.090	1.126	162805	9948	3269	7	59	11
ibm08	1.226	1.134	1.106	308106	18768	3084	16	75	14
ibm09	1.333	1.070	1.113	322633	20417	5905	15	66	14
ibm10	1.240	1.099	1.101	548402	25541	7261	20	96	20
ibm11	1.195	1.051	1.098	442739	22808	7902	13	92	21
ibm12	1.279	1.183	1.117	583444	97872	5774	30	102	21
ibm13	1.700	1.098	1.104	730804	69115	9984	37	130	25
ibm14	1.342	1.109	1.109	935023	97176	11153	65	296	69
ibm15[1]	4.431	1.145	1.107	10994359	297548	18233	312	359	76
ibm16	1.508	1.107	1.104	1401502	66903	16335	130	452	91
ibm17	1.843	1.090	1.067	3228739	352799	19938	210	428	97
ibm18	1.249	1.101	1.102	1889705	47661	15011	107	515	99
Average	1.326	1.102	1.106	1227000	71952	9816	55	157	33

significant improvements on wire length and stability. To the best of our knowledge, this is the first attempt using Delaunay triangulation to perform placement spreading. We believe there is still a lot of room to improve and other effects such as timing and wirelength to be incorporated.

Chapter 4

A New LP Based Incremental Timing Driven Placement

In previous chapter, the computational geometry based placement migration algorithm implicitly preserves the timing while removing cell overlaps. In this chapter, I present a LP based timing driven placement algorithm to reduce the delay on critical paths explicitly.

4.1 Introduction

The main focus of this chapter is to address the critical path improvement problem in high performance design. In a typical custom design flow for high performance microprocessors, the chip is floorplaned into functional regions, then hierarchically partitioned into basic design units. The size of the basic design unit is usually small, ranging from a few hundred gates to tens of thousands of gates. Even for these relatively small circuits, timing driven placement is very important since the gate delay is very sensitive to wire capacitance load and input slew in deep submicron technology. Therefore, cell placement in high performance designs often involves extensive manual tuning iterations to meet stringent timing requirement.

It has been reported that significant performance gap exists between ASIC

and custom design methodologies [24] because the custom designers understand the data flow of the circuit and take advantage of the inherent circuit regularity. On one hand, the ASIC methodology has fast turn-around time, but inferior performance for high-performance designs; on the other hand, the custom design methodology has much better performance, but very time-consuming.

To close such a gap, it is crucial to have powerful incremental timing driven placement which can iteratively improve the timing in custom designs. It helps to close not only the performance gap, but also the time-to-market gap.

Existing timing driven placement can be roughly classified into path-based and net-based approaches. The path-based algorithms try to minimize the critical paths of the circuit directly and have the advantage of holding an accurate timing view during the optimization. A common problem with them is their high computational complexity due to excessive number of paths. Path based timing driven placement includes [15, 31, 82, 46, 83, 51]. In [26], an accurate LP based differential timing analysis is proposed to improve the slack on critical paths that are identified by a static timer. However, one of the limitations of this approach is that if the static timer uses a sophisticated wireload model, such as a steiner routing tree based models, it is very difficult to formulate it into linear constraints. Therefore, any error arising from inaccurate wire models [26] in one stage of the path-based method will be propagated and accumulated in downstream stages on the timing path.

Net-based approaches usually transform timing to net budgets or weights, and perform constrained or weighted wire length optimization [61, 32, 38, 19, 39,

56, 25]. More recently, [76] proposed a sensitivity guided net weighting method that targets the net delay sensitivity. However, it did not consider slew propagation. The net-based approaches, especially the net weighting, have low computational complexity, high flexibility and is generally suitable for any wirelength minimization frameworks. Therefore, net-based approaches have more advantages as the circuit complexity continues to increase. However, net weighting often completely ignores slew propagation. Since timing is inherently path based, an effective net weighting algorithm should be based on path analysis and consider timing propagation. Furthermore, net-based approaches are often done in an ad-hoc manner and have problems with convergence [79, 51]. For instance, while the delay on critical paths decrease, other paths become critical, and this leads to a convergence problem. A systematic way of explicit perturbation control is important for net-based algorithms.

In this chapter, we present an LP-based incremental timing-driven placement optimizer. Our key contributions include:

- Our LP framework is net-based, but it takes advantage of the path-based delay sensitivity with limited-stage slew propagation. Thus it combines the advantage of net-based approach (flexibility/lower computational complexity) and path-based approach (more accurate timing view).

- Our LP formulation considers not only cells on the timing-critical paths, but also cells that are logically adjacent to the critical paths in a unified manner, through weighted LP objective function and net stretching bound constraints.

Therefore, our approach has precise control on timing perturbation during the optimization.

- We propose a timing aware spreading/legalization method to preserve timing for high performance custom designs. Our algorithm has been tested on a set of 65nm industry circuits from a multi-GHz microprocessor. It achieves much better timing even on carefully hand-tuned circuits (on average 20ps worst slack reduction, which is significant as the clock period is only a few hundred of pico-seconds)

The rest of the chapter is organized as follows: The problem formulation is in section 4.2. We discuss how to generate the path-based delay sensitivity net weights in section 4.3. In section 4.4, we show a method to construct the criticality adjacency network. The overall LP program is presented in section 4.5. Section 4.6 presents the timing aware spreading algorithm. Experimental results are shown in section 6.6. We summarize in section 6.7.

4.2 Problem Formulation

Table 4.1 lists the key notations used in the chapter.

4.2.1 LP formulation

In our algorithm, the timing optimizer selects a few critical paths from a timing report generated by an accurate static timer. Then it computes the delay propagation sensitivity on each net and inspects and classifies cells and nets into different categories based on their "criticality", which is logically how close they

Table 4.1: The key notations in this chapter.

c	The unit capacitance
r	The unit resistance
L_j	The wirelength of net j
Cap_j	Total output capacitive load on net e_j
$Cpin_j$	The sum of gate capacitance driven on net e_j
$Slew_i$	The input slew to cell i
Dg_i	The delay on cell i
Sg_i	The slew on cell i
K_D	Constant 0.69
K_S	Constant 2.2
a_i	The slew coefficient in cell i's delay formula in (6.4)
b_i	The delay coefficient in cell i's delay formula in (6.4)
u_i	The slew coefficient in cell i's slew formula in (6.5)
v_i	The delay coefficient in cell i's slew formula in (6.5)
De_j	The delay on net j
Se_j	The slew on net j
S_j	The delay propagation sensitivity of net j

relate to critical paths. As the linear program has a system of well developed theories to solve, we formulate the timing optimization problem into an LP problem and solve it optimally.

The half parameter bounding box wirelength (HPWL) model can be formulated exactly into an LP framework. Our algorithm uses HPWL for wirelength estimation and the linear gate delay and transition/slew models for delay computation. Although HPWL may not be well correlated with the final routed wire, it still captures the fidelity of the problem with reasonable accuracy. A carefully designed algorithm can take advantage of the accurate timing information generated by a static timer to achieve the optimization objective for a certain level of accuracy.

The objective of our algorithm is to minimize the delay on timing critical paths. We formulate the linear program to minimize the weighted wirelength on selected critical timing paths,

$$minimize \quad \sum_p \sum_j L_{p,j} S_{p,j} \tag{4.1}$$

where $L_{p,j}$ is the wirelength of net e_j on timing path p, and $S_{p,j}$ is the delay propagation sensitivity of net e_j. In the following sections, we formulate the models and constraints of the LP problem.

4.2.2 The capacitive load and delay models

For cell n_i, center coordinates x_i, y_i are the variables of the LP program. For a net e_j, To model HPWL, four variables l_j, r_j, t_j and b_j are used to represent left, right, top, and bottom locations of the bounding box of net e_j. Assuming k cells are connected to net e_j, we have

$$\begin{aligned} l_j &\leq x_i + pin_x(i,j) \\ r_j &\geq x_i + pin_x(i,j) \\ t_j &\leq y_i + pin_y(i,j) \\ b_j &\geq y_i + pin_y(i,j), \quad i = 1,2,..,k \end{aligned} \tag{4.2}$$

where $pin_x(i,j)$ and $pin_y(i,j)$ are the pin offset of cell i that connected to net e_j in horizontal and vertical directions respectively. The wirelength of net e_j is represented by L_j. We have

$$L_j = r_j - l_j + t_j - b_j \tag{4.3}$$

We use Cap_j to denote the total output capacitive load on net e_j. It is the sum of the wire capacitance of net e_j and the total pin capacitance driven on net e_j, which is denoted by $Cpin_j$, given by

$$Cap_j = c \cdot L_j + Cpin_j \tag{4.4}$$

where c is the unit capacitance constant. Assuming n_i is the driver of net e_j, the maximum capacitive load driven by n_i should not exceed the library specified maximum load $CMax_i$

$$Cap_j < CMax_i \tag{4.5}$$

To formulate the optimization problem into an LP program, we use the linear delay models for gates and the Elmore delays for wires [33]. The gate delay and transition are linear functions of input slew, *Slew*, and total capacitive load, *Cap*. We compute the fitting coefficients of the linear models based on a SPICE circuit simulation generated library. Note that the delay models for each pin of a gate, and for the falling or rising transition are different. We use different models for different pins and different transitions in the implementation and show only one formula here for simplicity. The gate delay Dg_i is given by

$$Dg_i = d_I + a_i \cdot Slew_i + b_i \cdot Cap_i \tag{4.6}$$

The gate transition Sg is given by

$$Sg_i = s_I + u_i \cdot Slew_i + v_i \cdot Cap_i \tag{4.7}$$

where a_i, b_i, u_i, and v_i are the fitting coefficients. d_I and s_I denote the intrinsic delay and slew of the corresponding pin of the cell.

The Elmore delay is used to estimate wire delay and slew on net e_j, which are given by

$$De_j = K_D \cdot r \cdot L_j \cdot (\frac{c \cdot L_j}{2} + Cpin_j) \tag{4.8}$$

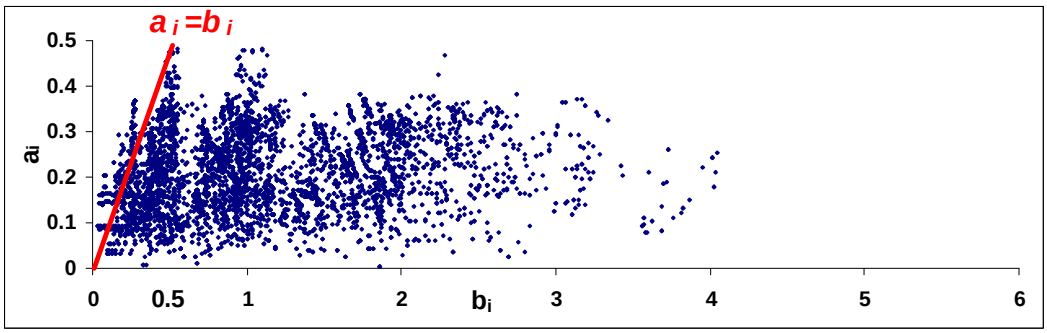
$$Se_j = K_S \cdot r \cdot L_j \cdot (\frac{c \cdot L_j}{2} + Cpin_j) \tag{4.9}$$

In recent publications, it has been shown that interconnect delay starts to dominate in deep submicron designs [37]. However, we should clarify that De_j in formula (4.8) is not the commonly referred interconnect delay, which is the part of the gate delay resulting from driving the interconnect/wire capacitance. Instead, De_j is the incremental RC delay on the wire, which is still relatively small for local nets under current technologies.

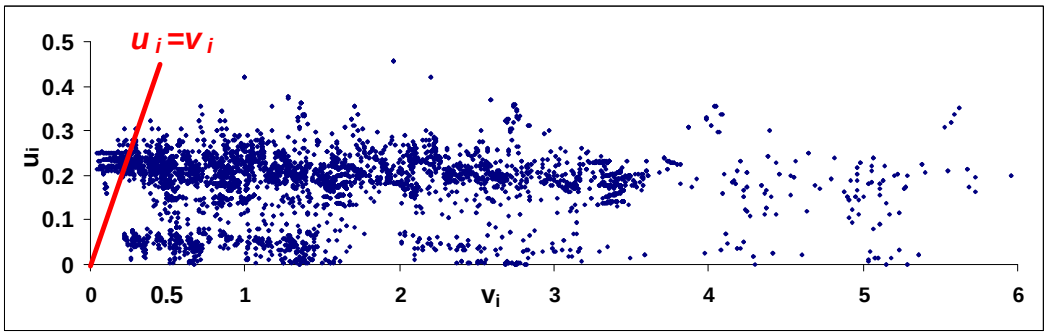
4.3 Path Based Delay Sensitivity

Any change on a net will affect the delay and slew of not only the driver gate, but also all downstream receiver gates, because the change of slew on a net will propagate. The delay propagation sensitivity of a net is a measurement of the sensitivity of the path delay to a wire length change, i.e., to estimate the change in path delay due to wire adjustments. An effective net weighting method should not only consider the current stage, but it should also have a path or global timing view embedded in the formulation.

Our limited-stage delay propagation sensitivity computation considers only two stages. We have the following observation from extensive experiments.



(a) The coefficient a_i and b_i in gate delay formula (6.4)



(b) The coefficient u_i and v_i in gate slew formula (6.5)

Figure 4.1: The normalized coefficient for *Slew* is much smaller than that for *Cap* in both formulas

on delay of the current stage than its receiving gates for most cases, because the gate delay and slew are more sensitive to the output capacitive load than to the input slew for the majority types of gates.

Figure 4.1 plots the normalized coefficients in formula (6.4) and (6.5) for all combinational gates in the library. In Figure 4.1(a), a point represents a pair of coefficients a_i and b_i in formula (6.4) for one gate . We see that the coefficient

corresponding to Cap_i is much larger than that for $Slew_i$ in both gate delay and slew formulas in most of cases. In other words, the delay and slew are more sensitive to the output capacitive load than to the input slew. The impact of a wirelength change on delay for the down stream gates is much smaller, and decreases quickly. Meanwhile, the inaccuracy of HPWL wire model still dominates; adding more stages may not help but will increase the computation complexity. Therefore, we limit the delay propagation sensitivity computation two stages.

We use the example in Figure 4.2 to show how to compute the delay propagation sensitivity. In Figure 4.2, cell n_i drives net e_j on a timing path, and cell n_{i+1} is the receiver gate connected to net e_j. Let S_j denotes the delay propagation sensitivity for net e_j. We have

$$S_j = \frac{\partial D_j}{\partial L_j} \tag{4.10}$$

where D_j is the portion of delay associated with net e_j. If the wirelength of net e_j changed by $\triangle L_j$, $\triangle D_j$ changes due to three components, the delta delay on the driving gate i, $\triangle Dg_i$, on net j, $\triangle De_j$, and on the receiving gate $i+1$, $\triangle Dg_{i+1}$.

$$\triangle D_j = \triangle Dg_i + \triangle De_j + \triangle Dg_{i+1} \tag{4.11}$$

From Equation (6.7), (6.4), (6.5), (4.8), and (4.9), we have

$$\begin{aligned} \triangle Dg_i &= b_i \cdot c \cdot \triangle L_j \\ \triangle De_i &= K_D \cdot r \cdot \triangle L_j \cdot (\frac{c \cdot \triangle L_j}{2} + Cpin_j) \\ \triangle Dg_{i+1} &= a_{i+1} \cdot (\triangle Sg_i + \triangle Se_j) \end{aligned}$$

where $\triangle Sg_i$ denotes the slew change on cell n_i, and $\triangle Se_j$ is the slew change on net e_j. We have

$$\begin{aligned} \triangle Sg_i &= v_i \cdot c \cdot \triangle L_j \\ \triangle Se_j &= K_S \cdot r \cdot \triangle L_j \cdot (\frac{c \cdot \triangle L_j}{2} + Cpin_{j+1}) \end{aligned}$$

The formula (4.10) becomes

$$\begin{aligned} S_j &= b_i \cdot c + K_D \cdot r \frac{c \cdot \triangle L_j}{2} + K_D \cdot r \cdot Cpin_j \\ &+ a_{i+1} \cdot (v_i \cdot c + K_S \cdot r \frac{c \cdot \triangle L_j}{2} + K_S \cdot r \cdot Cpin_{j+1}) \end{aligned} \tag{4.12}$$

$\triangle L_j \to 0$ gives

$$S_j = c \cdot (b_i + a_{i+1} \cdot v_i) + r \cdot (K_D \cdot Cpin_j + K_S \cdot a_{i+1} \cdot Cpin_{j+1}) \tag{4.13}$$

In above formula, the value of the unit resistance r is in order of magnitude smaller than that of the unit capacitance c, and the dominant term in formula (4.13) is $c \cdot (b_i + a_{i+1} \cdot v_i)$. Formula (4.13) is used to compute the delay propagation sensitivity of the net and also helps to guide the timing aware spreading/legalization.

4.4 Criticality Adjacency Network

To optimize the delay on critical paths, we adjust the coordinates of all cells associated with critical paths. If we do not control the timing perturbation on non-critical paths during the optimization, non-critical paths may become critical. In Figure 4.3(a), the path $n_1 \to A \to B \to n_2$ is critical, and the path $n_3 \to C \to B \to n_4$

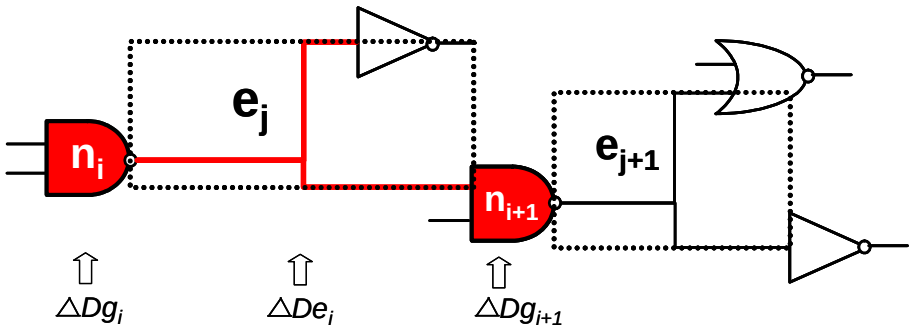
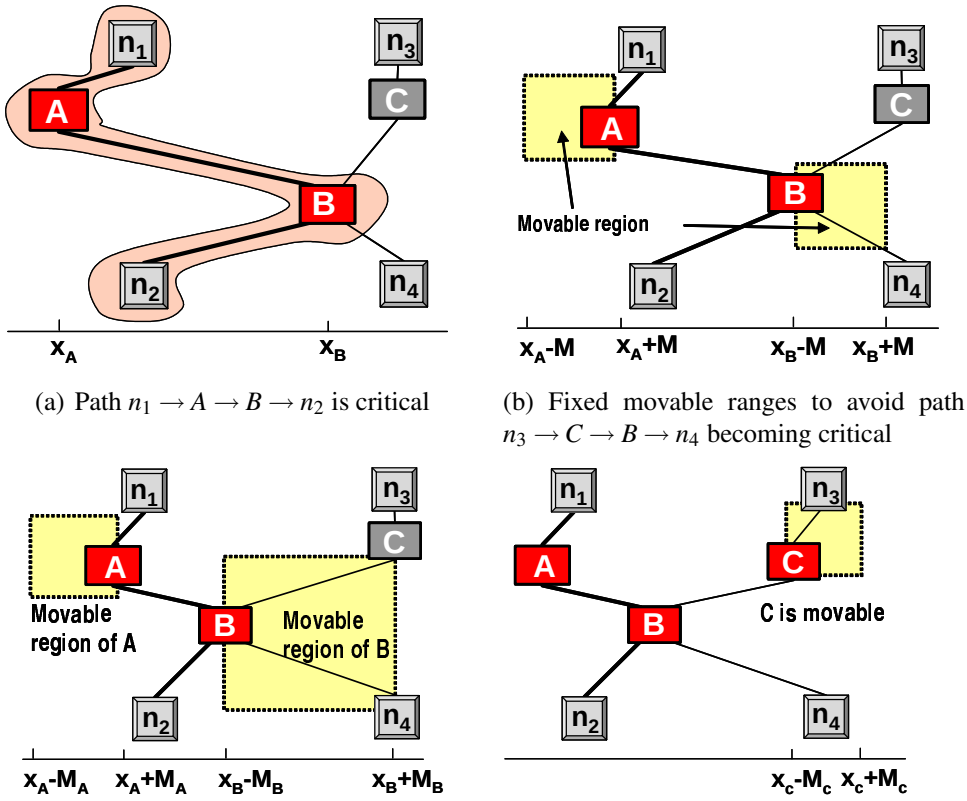


Figure 4.2: A circuit example for delay propagation sensitivity computation

may become critical after the optimization, as shown in Figure 4.3(b). Previous LP based approaches such as [39] [25] [26] set a fixed range to restrict every movable cell, as shown in Figure 4.3(b).

However, the delay of some cells may be extremely sensitive to wirelength changes, and other cells can be moved farther without significantly affecting the timing on non-critical paths. Such a potential to move is determined by not only the delay sensitivity of the net it drives, but also the "criticality" of the net and the cell itself. In other words, how sensitive the timing is subject to the net change and how logically "close" a cell is to the critical paths. In the following, we present the **criticality adjacency network** to classify cells and nets into different categories depending on their "criticality". As shown in Figure 4.3(c), we set different maximum movable ranges for cell A and B depending on the sensitivities of all nets they connected to. Furthermore, cell C in Figure 4.3 is not on critical path, but is also movable for it is logically adjacent to critical path, i.e., cell C is critical adjacent, as show in Figure 4.3(d).



(a) Path $n_1 \rightarrow A \rightarrow B \rightarrow n_2$ is critical

(b) Fixed movable ranges to avoid path $n_3 \rightarrow C \rightarrow B \rightarrow n_4$ becoming critical

(c) Variable movable ranges depend on the sensitivity of the connected nets

(d) Cell C becomes movable in the criticality adjacency network

Figure 4.3: The advantages of the criticality adjacency network

4.4.1 Criticality adjacency network

Figure 4.4 is an example of a combinational netlist with one critical path. Let G= (N, E) represents a netlist that has n cells, $N = \{n_1, n_2, ...n_n\}$, and m nets, $E = \{e_1, e_2, ...e_m\}$. The criteria to construct the criticality adjacency network is essentially based on how close the non-critical branches relate to critical paths and if they are helpful for reducing the delay on critical paths.

Let symbol $\rightarrow$ denotes the connection relationship. The construction of the criticality adjacency network is through the following definitions.

Definition 1. *$N(0)$ represents the set of cells on critical paths, and $E(0)$ is the set of nets on critical paths.*

$N(0)$= $\{n_1, n_2, n_3, n_4\}$ and $E(0) = \{e_4, e_5, e_6\}$ in Figure 4.4. All non-critical cells and nets in the circuit are classified by following definitions.

Definition 2. *$N(1)$ is the set of cells connected to nets in $E(0)$, excluding all critical nodes in $N(0)$. Set $N(2)$ contains all cells connected to cells in $N(0)$, excluding cells in $N(0)$ and $N(1)$. $N(3)$ is the set contains all other cells in N.*

Therefore,

$$\begin{aligned} N(1) &= \{c : c \rightarrow E(0), c \in N \setminus N(0)\} \\ N(2) &= \{c : c \rightarrow N(0), c \in N \setminus (N(0) \cup N(1))\} \\ N(3) &= \{c : c \in N \setminus (N(0) \cup N(1) \cup N(2))\} \end{aligned}$$

Hence, in Figure 4.4, $N(1) = \{n_5, n_6\}$, $N(2) = \{n_7, n_8, n_9\}$, and $N(3) = \{n_{10}\}$.

Definition 3. *$E(1)$ is the set of nets connected to cells in $N(0)$, excluding nets in $E(0)$. $E(2)$ is the set for nets connected to cells in $N(1)$ or $N(2)$, excluding nets in $E(0)$ and $E(1)$. All other nets are in set $E(3)$.*

Similarly,

$$
\begin{aligned}
E(1) &= \{e : e \rightarrow N(0), e \in E \setminus E(0)\} \\
E(2) &= \{e : e \rightarrow (N(1) \cup N(2)), e \in E \setminus (E(0) \cup E(1))\} \\
E(3) &= \{e : e \in E \setminus (E(0) \cup E(1) \cup E(2))\}
\end{aligned}
$$

In the example in Figure 4.4, $E(1) = \{e_8, e_{11}\}$, $E(2) = \{e_1, e_2, e_3, e_7, e_9, e_{10}, e_{12}, e_{13}\}$, and $E(3) = \{e_{14}, e_{15}\}$.

By classifying cells and nets based on "criticality", our algorithm optimally moves cells not only in $N(0)$ and $N(1)$, but also in $N(2)$. All cells in $N(3)$ and all nets in $E(3)$ are fixed. Therefore, the criticality adjacency network helps to obtain more room for optimization, while explicitly controls the timing perturbation.

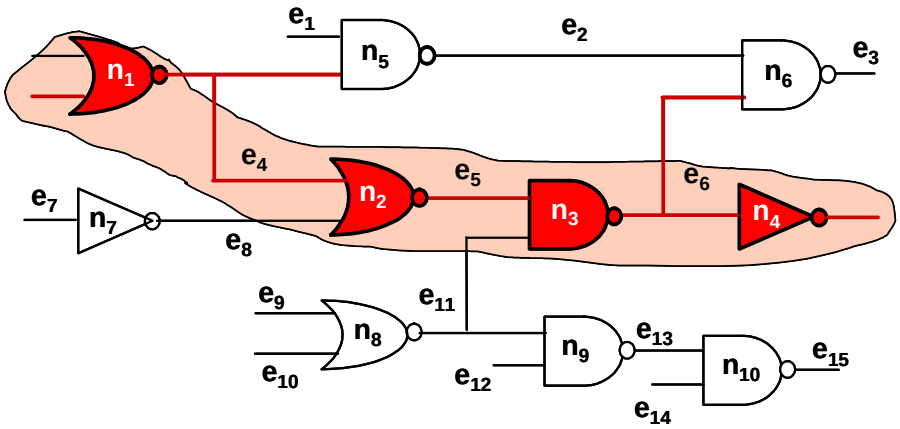


Figure 4.4: The criticality adjacency network

4.4.2 The timing perturbation constraints

With the help of the criticality adjacency network, instead of setting a fixed maximum movement range for all cells, we set a sensitivity based stretching bound for each net. The tightness of the stretching bound of a net is based on how sensitive the path delay is subject to the net change.

For a net e in $E(1)$ or $E(2)$, n_d is the driver cell and n_i is one of the receiver cells. From each pin of cell n_d to the receiver cell n_i, we compute a delay propagation sensitivity score. We compute the inverse of the sensitivity score and use it as the weight for net e. The sensitivity weights for all nets are scaled between zero and Max. Max is an experimental setting parameter. Let W_e denotes the weight for nets in $E(2)$ and $W_e^{'}$ denotes for nets in $E(1)$. The cell movement is restricted by the following net stretching bound constraints,

$$\begin{aligned} L_e - W_e \cdot L_e \leq L_e \leq L_e + W_e \cdot L_e, \forall e \in E(2) \\ L_e - W_e^{'} \cdot L_e \leq L_e \leq L_e + W_e^{'} \cdot L_e, \forall e \in E(1) \end{aligned} \tag{4.14}$$

The criticality adjacency network may be expanded to have more criticality levels to include more movable cells. Our experience shows that current level of critical adjacency network is sufficient.

4.5 The Overall Linear Program Algorithm

We formulate the optimization problem into an LP program and perform weighted critical nets optimization. Powered with the criticality adjacency network,

the algorithm unifies the objective of timing optimization and perturbation control into one LP framework.

Furthermore, the slack is also considered for weight adjustment. We adjust the weight of nets on critical paths according to their slacks or original delays. Assuming there are top P critical paths in a circuit, and the delay in the timing report for a path p is t_p. For a net e_j, it is possible that the net e_j belongs to P_j critical paths. Let S'_{k,e_j} denotes the delay propagation sensitivity of net e_j corresponding to the critical path p. Then, the adjusted delay propagation sensitivity weight of net e_j, which is denoted by S'_{e_j}, becomes

$$S'_{e_j} \geq S'_{p,e_j}, \quad p = 1,2,...,P_j \tag{4.15}$$

where,

$$\begin{aligned} S'_{p,e_j} &= \frac{t_p}{T_{min}} \cdot S_{p,e_j} \\ T_{min} &= min(t_p), \quad p = 1,2,..,P \end{aligned} \tag{4.16}$$

T_{min} is the shortest path delay among all P critical paths, which is used to normalize the original delay of all paths. Formula (4.16) adds additional weight for paths with larger negative slacks. Using the adjusted delay propagation sensitivity as net weight, the objective of the LP program is to minimize the sum of the weighted wirelength on critical paths. The following formulation is equivalent to formula (4.1).

$$\begin{aligned} &min \ \sum S'_{e_j} \cdot L_{e_j} \\ &\forall e_j \rightarrow E(0) \end{aligned} \tag{4.17}$$

The LP is formed under a set of constraints defined in previous sections. Such as the wire length and capacitive load constraints (4.2), (4.3), and (6.7), the maximum load constraint (4.5), and the timing perturbation constraints (4.14), etc.

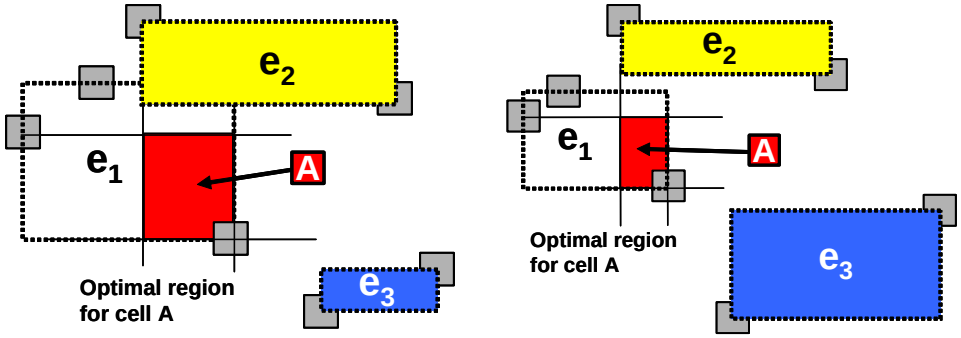
4.6 Timing aware spreading for legalization

The timing optimizer will generate a solution with cell overlaps. To legalize the placement, timing improvements may decrease. It is important that the legalization algorithm should avoid too much timing degeneration. According to [75] and [60], to maintain the relative orders among cells during the legalization helps preserve timing. We use the bin-stretching and Delaunay-triangulation based spreading algorithms similar to [60] for cell spreading to reduce overlaps.

We modify the spreading process in [60] to become timing aware to help cells with higher delay sensitivities move closer toward their "optimal" regions. The BoxPlace heuristic [54] is an effective method to reduce the wirelength in detailed placement. In brief, the BoxPlace moves a cell to the mean of its connected net bounding boxes to reduce the wire length. We propose a weighted BoxPlace to improve the cell spreading timing aware.

In Figure 4.5, e_1, e_2 and e_3 are nets connected to cell A. Figure 4.5(a) shows the optimal region for cell A, which is the medium of all cells connected to cell A. Figure 4.5(b) shows that nets are shrunk or expanded depending on their delay sensitivity weights. Conceptually, a net with higher delay propagation sensitivity will be shrunk and a net with lower weight will be expanded. The new optimal region for cell A shown in Figure 4.5(b) is better than that in Figure 4.5(a) from

the timing perspective. To pull cell A to its optimal region, a timing optimization force is generated on cell A. The timing force is scaled and vector combined with the spreading force to generate a timing aware cell spreading force to guide the movement of a cell. The cell spreading stops once the cells density distribution satisfies certain criteria, and we legalize the placement.



(a) The optimal region for cell A (b) The weighted optimal region for cell A

Figure 4.5: Under the weighted nets e_1, e_2 and e_3, the cell's optimal region changed

4.7 Experimental Results

We implement the algorithm in C++ and use the commercial tool MOSEK [64] as the LP solver. Seven circuits from a multi-GHz processor in 65nm process technology are used for experiments. The circuits are manually placed and have been optimized by designers to obtain the desired performance. The circuit sizes range from 6k standard cells to 28k. We take the critical paths above a certain threshold to optimize, then call the timing aware legalizer to remove overlaps. For each circuit, different thresholds are tested, and the best result is kept. Although

those circuits have been manually optimized, our algorithm still achieved significant improvements; the result is shown in Table 4.2.

In table 4.2, column *Gates* and *Nets* report the number of cells and nets of the testcases. Column *M* is the number of movable cells in the LP formulation. Column *Init* report the initial worst slack of the circuit. Column *Fin* reports the worst slack after the optimization. Column $-SL$ summarizes the worst slack improvement on each circuit. Column $-TNS$ is the total negative slack improved. Column *Base* is the timing baseline we used to measure how much slack improvement is obtained. If *Base* equals 0, $-TNS$ reports the total negative slack reduced. Because the final worst slack in a few circuits is positive, we use a *base* larger than zero to measure the total slack improvement for those circuits. Note that *ckt*1 has a positive initial worst slack. We should point out that it is meaningful to improve a positive worst slack design unit, because any slack improvement can be traded for power reduction later, where designer downsizes gates on timing paths with large positive slack to reduce the power consumption.

Column *OrgWL* and *FinWL* are the initial and final HPWL wirelength. $\triangle WL$ is the wirelength change. We can see that most of wirelength changes are within 0.1%, which implies that the disturbance to the circuits is very small. We did not show the computation timing data because the computation is very fast for all testcases. The algorithm only handles a small number of cells and because of the efficient linear formulations, the algorithm runs very fast, within a few minutes for the largest circuit. By moving a small number of critical cells, slack can be improved considerately. We observe a slack improvement of 20 picosecond on aver-

age, which is significant considering the circuits have already been hand-optimized.

Table 4.2: Experimental results

	Gates	*Nets*	*M*	*Init*	*Fin*	$-SL$	$-NS$	*Base*	*OrgWL*	*FinWL*	$\triangle WL$
ckt1	6671	7261	340	22	32	10	297	40	193904	192962	-0.50%
ckt2	8249	9640	89	-15	5	20	412	10	240671	239693	-0.41%
ckt3	9541	12161	92	-33	-15	18	937	0	267661	268240	0.22%
ckt4	13220	14479	164	-54	-30	24	559	0	483479	483037	-0.09%
ckt5	15486	19515	587	-37	-12	25	331	0	432319	436170	0.89%
ckt6	27014	28961	60	-3	12	15	136	15	659269	659171	-0.01%
ckt7	28535	31893	62	-36	-22	14	1211	0	921118	921518	0.04%

4.8 Summary

We proposed a new LP-based incremental timing optimizer for timing optimization in placement for high performance custom designs and ASICs. Our LP framework uses an accurate delay sensitivity based net-weighting method that combines the advantage of the path-based approach. We further presented a novel criticality adjacency network concept to formulate cells both on and adjacent to critical paths into the optimization framework, which helps to precisely control the timing perturbation during the optimization. In addition, we developed a timing aware spreading method to preserve timing during the legalization. Our experimental results showed that the proposed algorithm significantly improved timing on a set of manually-optimized industry circuits from a 65nm multi-GHz high performance custom processor.

Chapter 5

Pyramids: Computational Geometry-based Approach for Timing-Driven Placement

The mathematical programming based timing driven placements normally work on a set of critical paths and are expensive to run many iterations. From the experience of wire length driven placement, efficient techniques dealing with individual cell could be very effective. In chapter, I present a light-weight timing optimization technique that is shown effective on incremental timing optimizations, and potentially suitable for timing driven global placement.

5.1 Introduction

Global placement is a well-studied optimization that seeks to find a location for every cell in the design, typically via a wire length objective. Of course, as part of a physical synthesis flow, one also needs to satisfy the timing constraints. Despite the best efforts, timing-driven global placement will never be a complete solution, since one cannot glean an accurate picture of the current timing until the placement has stabilized. Thus, there is a need for tools that can take an existing placement and incrementally modify it to improve its timing characteristics.

This problem has been exacerbated by technology scaling, since the increase

in interconnect delay relative to gate delay means that cells placed far from their ideal timing locations suffer a greater timing penalty than for previous technology generations. It is made even worse by the emergence of multi-cycle which require careful latch placement to ensure balance of slacks for paths on both sides of the latch. Timing-driven global placement commonly uses net-weighting based methods to address timing [56, 76], but they are again inadequate to solve the problem completely. Thus there is a body of work mostly using mathematical programming to incrementally improve circuit timing [25, 26, 58]. Mathematical programming based approach is normally expensive in computation.

This work presents a new physical optimization technique called Pyramids, designed to incrementally improve the timing characteristics of a layout via cell movement. The name Pyramids comes the fact that one can solve for the optimal set of locations by finding the intersection of a set of pyramid shaped delay surfaces emanating from the cells incident to the cell of interest. This technique is simpler and faster than linear programming and runs in constant time. It can also be used in several ways, two of which we describe in this chapter. We make the following contributions.

- We show how the Pyramids solver can be used to perform timing-driven detailed placement via a bounding box capacitance model. Once can iterate over critical cells in the design and move them to better locations very efficiently. This is very effective for early timing-improvement of a global placer and could be embedded within a timing-driven placement algorithm.

- Later in a physical synthesis flow, one may require more accuracy and care to improve path delay. In this context we show how Pyramids can be used as critical path optimization under a linear delay model.

- Experiments show how effective these techniques are. The Pyramids based timing-driven detailed placer improves slack by 30.4% on average after optimized by the timing driven placer in Cadence SOC encounter. For a large commercial ASIC, our second critical path optimization technique improves slack by more than 40% of the cycle time.

These approaches present two applications of this solving technique, though it has other potential applications for modeling incremental timing optimization techniques.

5.2 Preliminaries of Pyramids formulations

The Pyramids can be applied in different stages of the physical synthesis flow. In earlier stages of the flow, such as the timing-driven global- and detailed-placement, where nets have not been optimally buffered, Pyramids uses the bounding box net model to estimate the capacitive load. In later stages of a physical synthesis flow, many buffers are added to decouple high fan-out nets. The Pyramids model decomposes the multi-terminal net into 2-pin timing arcs. We formulate the boundingbox-model-based Pyramids timing-driven placement (Pyramids-DP) in section 5.3, and timing arc model based Pyramids timing refinement algorithm (Pyramids-CP) in section 5.4.

The Pyramids algorithm identifies the movable gate m and constructs a subcircuit. A subcircuit G is a graph with the movable gate m and its connected gates, as wells all signal nets connecting the gates. The problem is to maximize the minimum slack of the subcircuit by moving m toward an optimal location. The static timing analysis is the basis of Pyramids timing optimization.

For a gate g_i in subcircuit G, let x_i, y_i denote the center coordinates of g_i. Let L_j denote the Half Parameter Wire Length (HPWL) of net n_j. In the rest of the chapter, the pin offset is not discussed for simplicity, and the pin offset has to be considered in implementation.

The unit capacitance constant is denoted by c, and the unit resistance constant is denoted by r. Let Cap_j denote the total output capacitive load on net n_j. It is the sum of the wire capacitance of net n_j and the total pin capacitance driven by net n_j, which is denoted by $Cpin_j$. Therefore, $Cap_j = cL_j + Cpin_j$.

The Required Arrival Time (RAT) on the inputs of a combinational gate g_i is the minimum of the required arrival time propagated back from the gates driven by g_i.

$$RAT_i = \min_{0 \le j \le p} \{RAT_j - rcL_j - Dg_i\} \tag{5.1}$$

The Actual Arrival Time (AAT) on the output of a combinational gate g_i is the maximum of the actual arrival time propagated from the drivers of g_i.

$$AAT_i = \max_{0 \le j \le k} \{AAT_{i_j} + rcL_j + Dg_i\} \tag{5.2}$$

The slack of net n_j is denoted by SL_j, which is the difference between the

required arrival time and actual arrival time

$$SL_j = RAT_j - AAT_j \tag{5.3}$$

The required arrival time and actual arrival time on the gates are generated by a static timer. The delay of a gate g is denoted by Dg. The gate delay is the linear functions of the input slew, $Slew$, and total capacitive load, Cap. The coefficients are computed by fitting the look-up table based standard cell timing library. Here we show a simplified form of the delay equation that does reflect the difference between pins. We use different models for different pins in the implementation and the worst case model from the falling and rising transitions.

The gate delay Dg_i is given by

$$Dg_i = d_I + a_i \cdot Slew_i + b_i \cdot Cap_i \tag{5.4}$$

where a_i and b_i are the fitting coefficients. d_I denotes the intrinsic delay of the corresponding pin of the gate. We use a static input slew for delay computation, which is generated by a static timer.

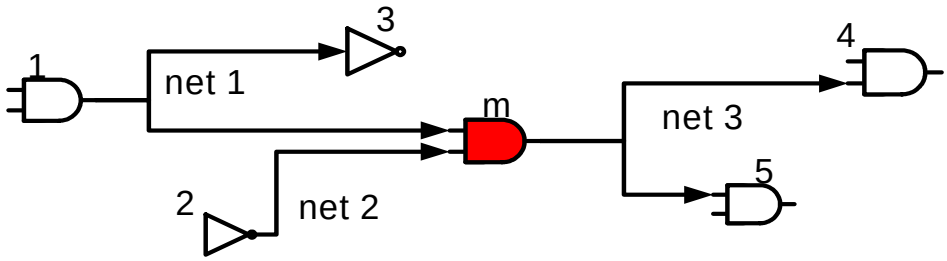


Figure 5.1: A subcircuit with one movable gate

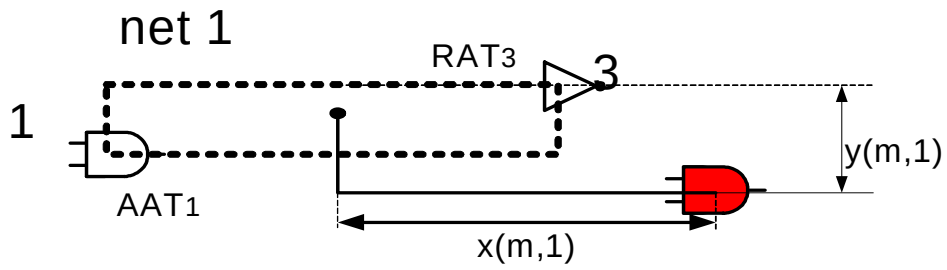


Figure 5.2: The definition of the net length on net 1

5.3 Pyramids algorithm for timing-driven detailed placement (DP)

The ideal of Pyramids algorithm is to transform a timing optimization problem into a geometric optimization problem. We refer the Pyramids algorithm for timing-driven detailed placement as Pyramids-DP in later sections.

Figure 5.1 is a simple subcircuit to illustrate the major steps in Pyramids-DP. In Figure 5.1, m is a movable gate, the movable gate could be a combinational or a sequential gate. All other gates connected with m are fixed. For all gates in the subcircuit, the slew rate, required arrival time and actual arrival time have been computed by a static timer. Assuming a net j is bounded by gate p and gate q on x dimension. Let $x_{(m,j)}$ denote the horizontal distance between m and the geometric center of net j, excluding m. We have

$$x_{(m,j)} = |x_m - 0.5(x_p + x_q)| \tag{5.5}$$

In Figure 5.1, net 1 and net 3 are the multi-pin nets and net 2 is a two-pin net. Figure 5.2 shows the bounding box of the net 1 with m detached. For net 1 in Figure 5.2, $x_{(m,1)} = |x_m - 0.5(x_1 + x_2)|$.

Let $y_{m,j}$ denote the vertical HPWL of net j. For net 1, $y_(m,1)$ is shown in Figure 5.2. We assume that gate *m* moves on x dimension only. In other words, $y_{(m,j)}$ is a constant in the formulation.

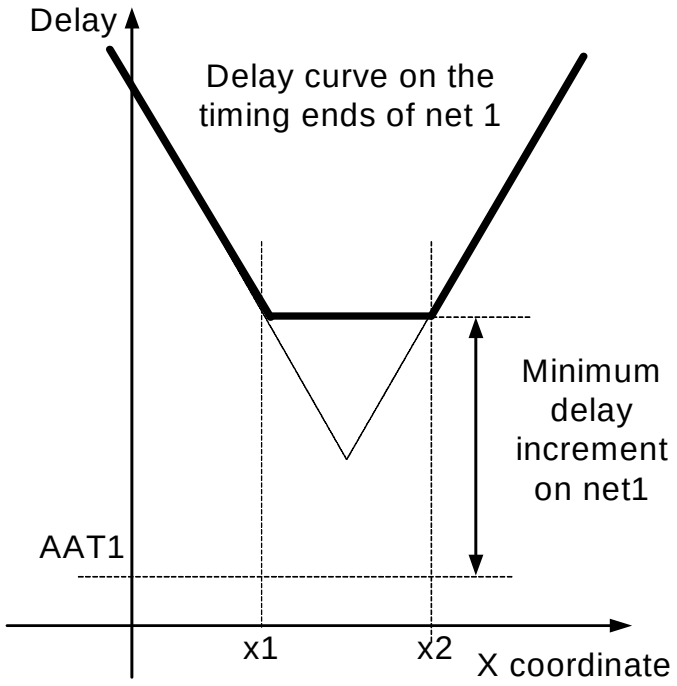


Figure 5.3: Delay curve on net 1

5.3.1 Optimal cell location computation in Pyramids DP

The relationship between the gate m's *x* location and the delay can be plotted on a two-dimensional space. The delay curve on the timing end of net j, which is also the arrival time on the input of gate *m*, is determined by the intersection of the

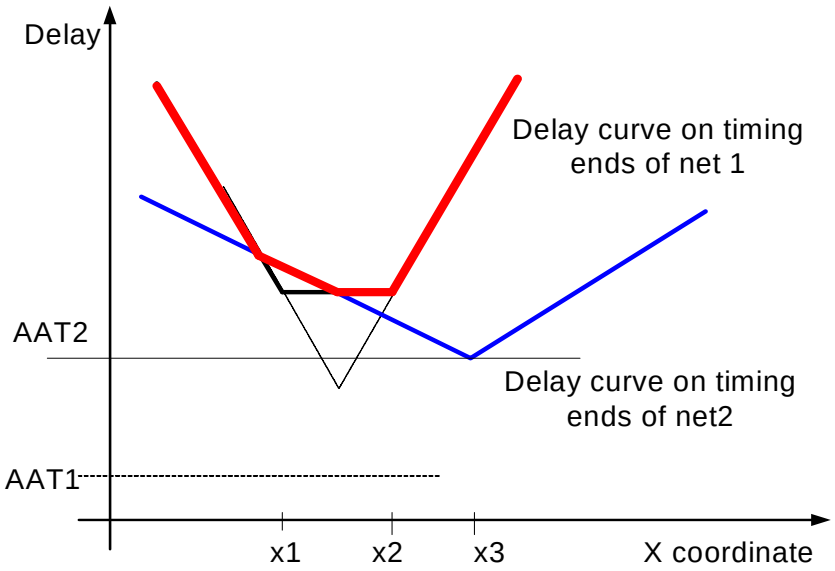


Figure 5.4: Delay curve on on inputs of gate m

following set of linear equations.

$$D_m = AAT_j + (a_j + r*c)(y_{(m,j)} + x_q - x_p) \quad (5.6)$$

$$D_m = AAT_j + (a_j + r*c)(y_{(m,j)} + 0.5(x_q - x_p)) \quad (5.7)$$

$$+(a_1 + rc)x_{(m,j)} \quad (5.8)$$

Let the vertical axis represent delay and the horizontal axis represent the x coordinate. Figure 5.3 plots the delay on net 1 as a function of the x location of m. x_1 and x_2 in Figure 5.3 denote the x coordinate of gate 1 and 2. AAT_1 is the actual arrival time on the output pin of gate 1, which is the driver of net 1. If gate m moves on the x dimension, based on Equation (5.2) and (5.4), the delay on net 1 is the maximum of the linear delay functions defined in Equation (5.3), which is plotted in Figure 5.3. Note that the slope of the delay lines are $\pm(a_1 + rc)$, where a_1 is the coefficient

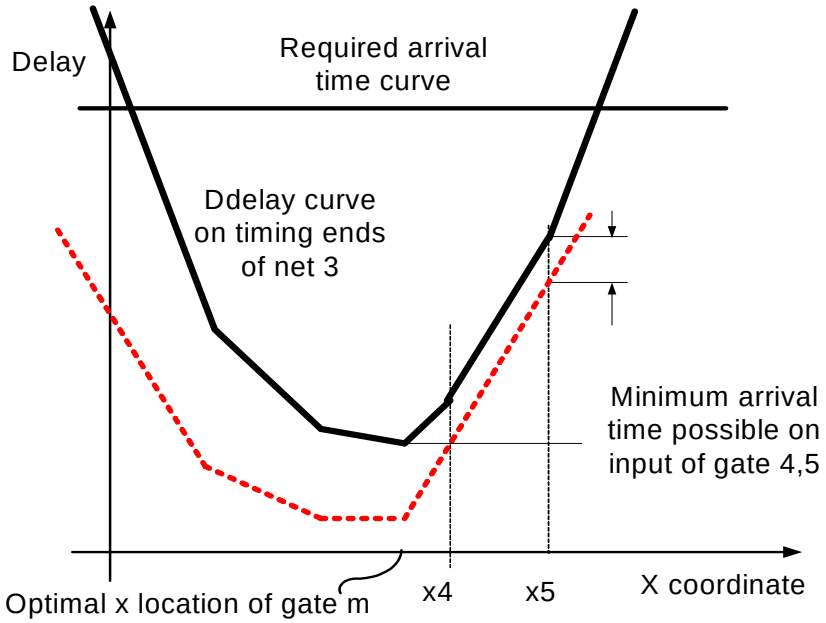


Figure 5.5: Delay curve on on net 3

in Equation (5.4).

According to Equation (5.2), the actual arrival time on the output of the combinational gate m is the sum of the maximum of the arrival arrival times on both inputs of m and the gate delay on m. The delay curve on the timing end of net 2 is the V-shape curve, as in Figure 5.4. Take the maximum of the V-shape curve and the delay curve on net 1, we have the line segments that represent the actual arrival time on the input of m, shown as the thick line segments in Figure 5.4.

Figure 5.3, 5.4, and 5.5 show the arrival time curve generation on the input of gate 3 and gate 4. x_4 and x_5 are the x coordinates of gate 4 and 5. The arrival time on the timing end of net 3 is the summation of the worst actual arrival time on

the inputs of gate m, the gate delay on m, and the net delay on net 3. Both the gate delay and net delay are linear functions of x_m. Therefore, we add up the delays on gate and net to the worst input arrival time on gate m to generate the arrival time curve on net 3, shown as the thick line segments in Figure 5.4. The horizontal line in Figure 5.4, represents the worst case required arrival time on the corresponding input of gate 3 and 4. The difference between the required arrival time and the actual arrival time curve is the slack curve, and the top of the slack curve is the best slack achievable by moving gate m. In this example, the optimal x location is shown in Figure 5.5, given a specified y location.

5.3.2 Pyramids-DP algorithm

In the following we discuss the application of Pyramids algorithm in timing driven detailed placement. The input of Pyramids is a legal placement. The placement is analyzed by a static timing analysis tool and the timing information is annotated in the placement database. The Pyramids-DP collects a few critical gates and sorts them by the worst slack on the pins. Pyramids starts with the worst gate and works on each gate in the queue order. Once a gate is moved the timing information is incrementally updated. Above process is repeated to a desired extent.

In each optimization iteration. Once a movable gate is selected, Pyramids analyzes the subcircuit and reject the gate if the bounding box of the the subcircuit is too small, which implies that it is no space for improvement. The bounding box of the candidate subcircuit covers a set of rows. Starting from the middle row of the subcircuit bounding box, the movable gate is assigned a row location, and Pyramids

computes the optimal x location, as shown in Figure 5.6. Then a row up and a row down is selected to compute the optimal location, and all slack improvement score is recorded. It is to be noted that the slack improvement score is concave on the row location. Pyramids does not need to compute the score of all rows. And during the experiments, the optimal x location is the same for each row on most of subcircuit.

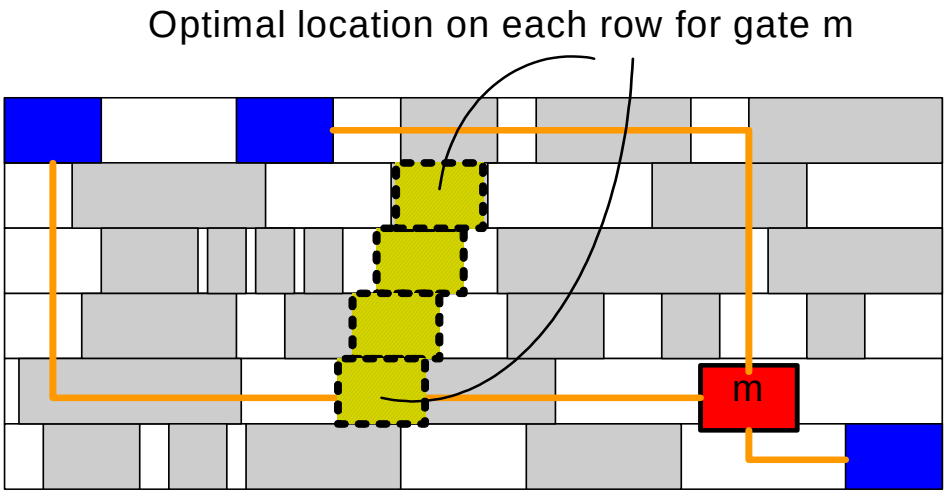


Figure 5.6: Compute optimal location for the movable gate on each row

Each row of the placement is divided into bins. The bin width is about 4 gate's average width, and bin density is computed. The top row is selected and if there is space, the movable cell is placed. Otherwise, the overlapped gate is tentatively move to a neighboring bin with lowest density. If the worst slack of that gate becomes negative, such a swapping is rejected and next row candidate is considered. Otherwise the gate movement is accepted. An simplified internal timer computes the slack changes during the swapping. The algorithm is shown in Algorithm 5. All gate are within row when the timing optimization procedure

terminates. There will be a small amount of residual overlaps in the x dimension that will be removed.

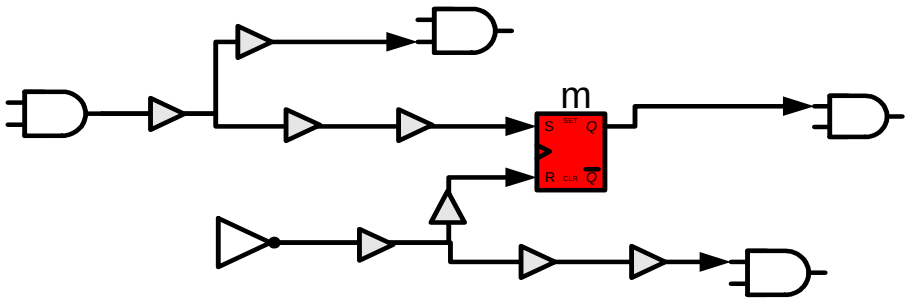
Algorithm 5 Pyramids-DP(Timing driven detailed placement flow)

```
1:  PyramidsDP(critical gates Queue)
2:  foreach gate in Queue
3:      Build the subcircuit, check if the subcircuit can be improved
4:      foreach row in candidate rows
5:          slackScore = ComputeOptimalLocation(subcircuit, row)
6:      end foreach
7:      Sort rows on slack improvement score
8:      foreach row in the sorted rows
9:          Move the overlap gate to the low density neighbor bin
10:         if Slack is negative, accept change break
11:     end foreach
12: end foreach
13: ComputeOptimalLocation(subcircuit, row)
14:     Compute the delay lines on inputs of the gate
15:     if (movable is a latch) then
16:         Compute the delay planes on all inputs
17:     else
18:         Intersect input delay lines to find the top line segments
19:         Grow output delay lines on top of input delay line segments
20:     end if
21:     Compute all slack lines based on the RAT
22:     Intersect all slack lines and find the bottom line segments
23:     Mapping the top point of the line segments to x-axis
24:     end if
25:     return optimal x and min-slack
```

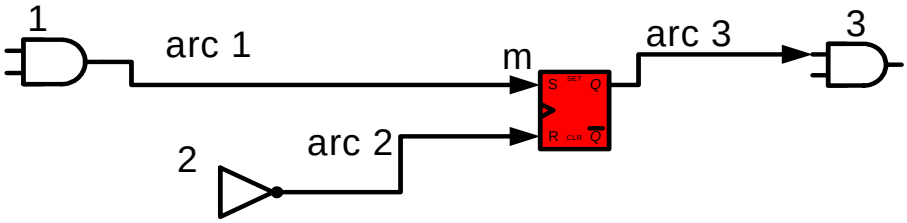
5.4 Pyramids algorithm for critical paths refinement (CP)

During later stages in a physical synthesis flow, Pyramids moves a few gates, especially the imbalanced latches, to further improve timing. We refer the Pyramids algorithm for critical paths refinement as Pyramids-CP in the following. Imbalanced latch is the latch with positive slack on one side and negative slack on the other side. Obviously, the improvement of imbalanced latches has the potential to improve timing. However, in later physical synthesis stages, many buffers are added to decompose high fan-out nets and linearize the delay on long nets. The bounding box net model for the multi-pin nets is no longer suitable.

Furthermore, the buffers inserted restrict the movement of gates. Whenever a gate is moved, the connected buffers has to be treated properly to avoid timing degradation. In the example shown in Figure 5.4. Assuming to move *m* closer to the input drivers will reduce the negative slack. The ideal way is to move the latch and buffer the nets simultaneously. However, multi-objective optimizations are often computational prohibitive and non-trivial to realize. Pyramids-CP computes the optimal location of the movable gate based on the timing estimation considering buffers that will be inserted in the future. Once the gate is moved, all the nets associated are buffered again. Such an approach has been proved effective and efficient by experiments, and is easily to integrity into a typical physical synthesis flow. To accommodate this approach, the following timing model os critical for Pyramids based timing refinement.



(a) Subcircuit with buffers



(b) New timing model

Figure 5.7: The net model in Pyramids-CP

5.4.1 Linear Buffered-Path Delay Estimation

Buffering can not be ignored during the later physical synthesis stages for interconnect delay estimation [29, 71, 7]. Therefore, supported with the buffering technology, the interconnect delay model must be aware of the buffers, which is going to be inserted in the future. We found that the linear delay model [70, 7] is best suited. Therefore all multi-pin nets are break down into timing arcs. As shown in Figure 5.7, on directly related timing arcs are included in the subcircuit. In this model, the delay along an optimally buffered interconnect is

$$delay(L) \quad = \quad L(cR_b + rC_b + \sqrt{2R_bC_brc}) \tag{5.9}$$

where L is the length of a 2-pin buffered net, R_b and C_b is the intrinsic resistance and input capacitance of buffers and gates while r and c are unit wire resistance and capacitance respectively.

Empirical results in [7] indicate that Equation(5.9) is accurate up to 0.5% when at least one buffer is inserted along the net. Furthermore, our own empirical results suggests the model is reasonable enough to justify the latch location. The details are described in section 5.5.2.

5.4.2 Pyramids-CP formulations

In this formulation, τ is a technology dependent parameter that is equal to the ratio of the delay of an optimally-buffered, arbitrarily-long wire segment to its length

$$\tau = delay(wire)/length(wire) \tag{5.10}$$

A timing arc is specified for a given net n driven by gate u and having sink v as $n_{u,v}$.

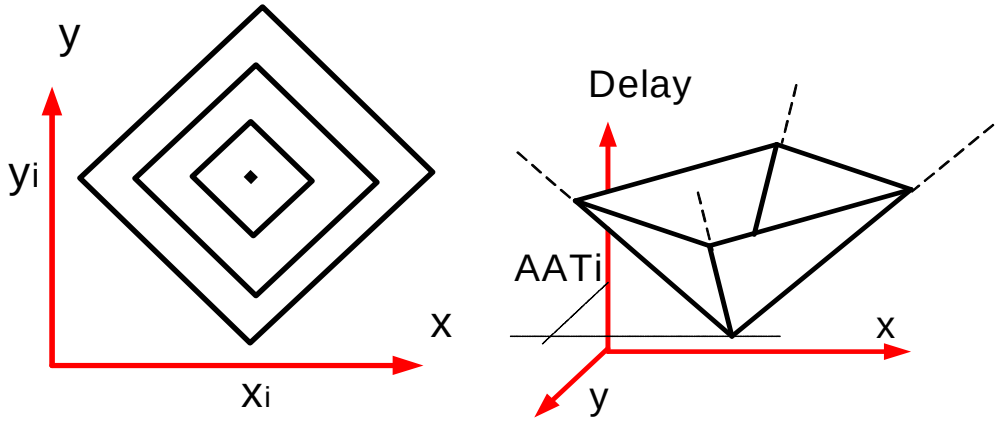
In Pyramids-CP, the definition of the required arrival time on a combinational gate g_i is similar as that in equation (5.1), with a different format.

$$RAT_i^{'} = \min_{0 \le j \le p} \{RAT_j - \tau \mathrm{L}_j - Dg^{'}\} \tag{5.11}$$

The actual arrival time on the output of a combinational gate g_i is

$$AAT_i^{'} = \max_{0 \le j \le k} \{AAT_{i_j} + \tau \mathrm{L}_j + Dg^{'}\} \tag{5.12}$$

where $Dg^{'}$ is not dependent on the capacitive load here.



(a) Manhattan contours (b) Delay surface: an upside down Pyramid

Figure 5.8: The shape of delay surface in 2d and 3d spaces

5.4.3 Compute the optimal location in Pyramids-CP

The required arrival time, actual arrival time and slacks are linear equations of the cell coordinate and can be plotted in a geometric space and solve by an geometric based approach. Assuming a gate moves on x and y dimensions simultaneously. The arrival time, or delay on a timing arc is a linear equation of the manhattan distance between driver gate and receiver gate of a timer arc, which are the intersection of planes if plotted in a 3D space. Let (x_i, y_i) denotes the coordinate of gate i. For a timing arc driving by gate i. The delay on the timing is determined by following equations.

$$Delay = AAT_i + \tau(|x_m - x_i| + |y_m - y_i|) \tag{5.13}$$

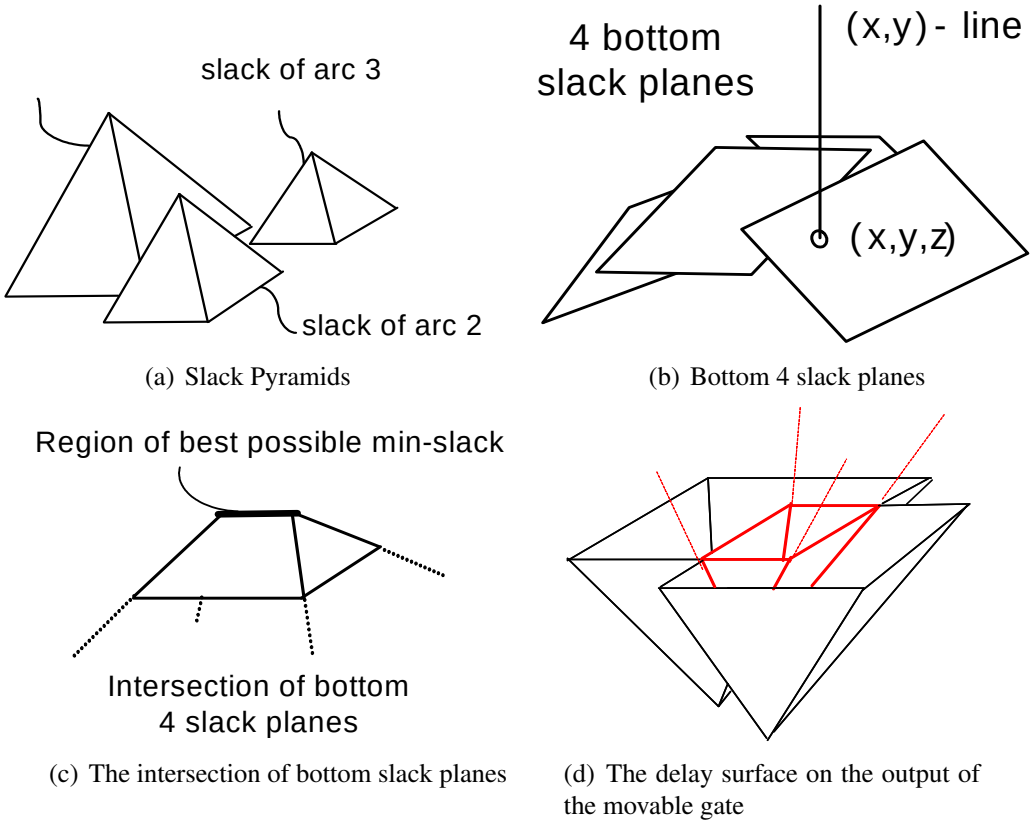
Above equation represents 4 planes in 3D space intersecting with each other. The 4 planes intersected into a shape of an upside-down pyramid in 3d space. We named

our algorithms Pyramids, which is after the shape of the delay surface. To visualize the concept, the contours of the pyramid delay surface is shown in Figure 5.8(a), and the shape of the delay surfaces are shown in Figure 5.8(b).

5.4.3.1 Optimize the sequential gate

If the movable is a sequential gate, as the case in Figure 5.7, the required arrival time on timing arcs are a constant. In other words, the require arrival time is a plane parallel to the surface of the x-y plane in 3d space. The difference between the required arrival time and the arrival time surface is the slack surface, which is in the shape of a pyramid. In this example in Figure 5.7, there will be totally 12 slack surfaces generated for timing arc 1, 2,and 3, as shown in Figure 5.9(a). There are 4 planes for each slack pyramid, and all slack surfaces can be categorized into 4 groups. The slack surfaces in the same group are parallel to each other.

To find the best slack surface possible, two testing points, such as $(0,0)$ and $(0, chip - y - dimension)$ are sufficient to find the orderings of all slacks surfaces. The idea is illustrated in Figure 5.9(b), and there will be 4 bottom slack planes. The intersection of 4 bottom planes form a "trough" shape polyhedron, which is the best possible worst slack region of the subcircuit, which is shown in Figure 5.9(c). Mapping of the top line segment of the trough to the placement region is the optimal region for the movable. To move the latch to any point on the line segment achieves the best possible worst slack.



(a) Slack Pyramids

(b) Bottom 4 slack planes

(c) The intersection of bottom slack planes

(d) The delay surface on the output of the movable gate

Figure 5.9: Computation of the optimal region in Pyramids-CP

5.4.3.2 Optimize the combinational gate

If the movable is a combinational gate, the actual arrival time of on the output of the gate is the maximum of the actual arrival times on the inputs of the movable with the increment of the gate delay. The max operation of the delay surfaces can be computed in similar way as the method to compute the bottom slack planes in section 5.4.3.1. Again, two testing points are sufficient to find the top 4 delay planes, as shown in Figure 5.9(d).

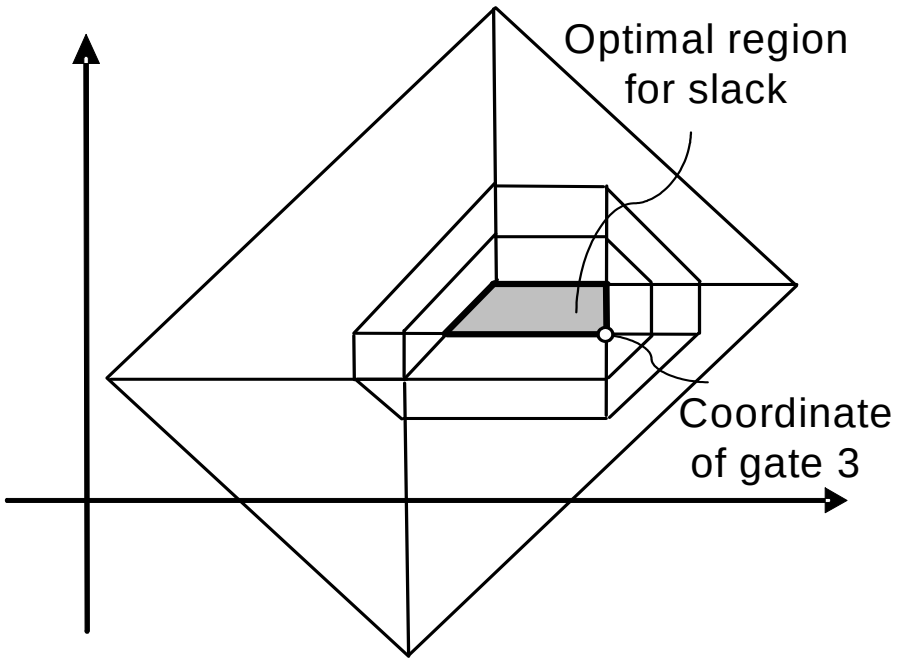


Figure 5.10: Delay surface contour on the output net

The delay pyramid of each output of the movable gate "grows" on top of the actual arrival time trough of the movable, which is a basin-type shape having nine delay planes with the bottom plane parallel to the x-y plane. The contours of the delay surfaces are shown in Figure 5.10. As the required arrival time on each output timing arcs is a constant plane parallel to the x-y plane. The bottom slack planes form a reverse basin similar as that formed by the delay planes. It is to be noted that the 9 bottom planes are required only for finding the entire optimal region. The implementation and computation is simplified if just one optimal point or a portion of the optimal region are needed, as the the optimal region intersects with bottom of the input delay trough.

5.4.4 Pyramids-CP algorithm

In this subsection, we discuss Pyramids-CP for critical path refinement, which mainly focus on improving the poorly placed latches. Once an imbalanced latch is selected, Pyramids select the subcircuit, and ripping off all buffers previous inserted. A snapshot of the previous is cached, which includes previous buffer locations, signal polarities, and the latch location. The Pyramids-CP improves the timing in the ”Do no harm” philosophy. In the Pyramids framework, nets are re-buffered after the latches movement, and the actual impact of the latch movement is evaluated by an incremental timer. If the timing is not improved due to due to varies reasons, such as blockages, buffering issues, etc., the changes are rejected and previous status of the subcircuit is restored.

The Pyramids-CP algorithm is shown in Algorithm 6.

5.5 Experiments

We have implemented Pyramids-DP and Pyramids-CP in C++. Pyramids-DP is based on the OpenAccess (OA) [69, 89], which is an open source database system for EDA applications. Pyramids-CP is implemented within an industrial physical synthesis flow. For Pyramids-DP, We have adopted the industrial strength, open source, static timing engine in the OA Gear package [68], as the timing analysis tool. There is a hypothetical standard cell library provided in OAGear to resemble a typical 180nm process. We have linearly scaled the 180nm library down to 130nm, 90nm, 65nm, and 45nm. For each technology node, the library parameters

Algorithm 6 The Pyramids-CP(For critical paths refinement

```
1:  PyramidsAlgorithmII(movable gate)
2:     Identify the subcircuit
3:     Rip-up buffers and save buffers in cache
4:     Compute the delay planes on inputs of the gate
5:     if (movable is a latch) then
6:        Compute the delay planes on all inputs
7:     else
8:        Find the top delay planes by ordering all input delay planes
9:        Grow output delay planes on top of input delay trough
10:    end if
11:    Compute all slack planes based on the RAT and delay planes
12:    Find the bottom slack planes by ordering all slack planes
13:    Find the optimal region by intersecting the bottom slack planes
14:    Move the gate a suitable optimal location
15:    Re-buffer the subcircuit
16:    Evaluate the subcircuit timing by incremental timing analysis
17:    if Timing is improved
18:       Keep the change, release the cache
19:    else
20:    Drop the change, restore the subcircuit from the cache
21:    end if
```

are scaled by a 80% to its previous technology generation. We use the 45nm library as the timing library for all OA based experiments. For benchmarks, we select the largest circuits in the ISCAS89 sequential logic benchmarks, and convert the netlist into OA format. The characteristics of the benchmarks are shown in Table 5.1.

5.5.1 Pyramids-DP experiments in OpenAccess environment

We tested Pyramids-DP on all circuits in Table 5.1. We use Cadence 2005 SOC encounter timing driven placer to generate the initial placements. The OAGear Timer is used to measure the worst case delay of the placement generated by Ca-

design	*cells*	*Regs*	*Pins*	*nets*
s838_1	404	32	37	404
s1196	513	18	30	500
s1238	616	18	30	603
s1423	631	74	24	627
s1488	665	6	29	647
s1494	672	6	29	654
s15850	788	165	38	701
s9234_1	1051	145	67	1006
s13207	1373	333	76	1235
s5378	1380	163	83	1332
s38584	7016	1178	233	6741
s35932	7630	1728	69	7311
s38417	8414	1564	136	8309

Table 5.1: The characteristics of the ISCAS benchmarks

dence timing driven placer. For each circuit, the timing target is set to a value about 95% of initial worst case delay. Pyramids-DP reads the initial placement and improves the timing further. Table 5.2 reports the further improvement accomplished by Pyramids-DP. Column 8 shows the target timing for each circuit. Column 2-4 report the worst negative slack (WNS), total negative slacks (TNS) and the wire length (WL) of the initial placements. Column 5-6 report the same of the Pyramids-DP improved results. Column 9-11 report the Pyramids-DP improvement by percentage. Compared with the inial placement optimized by Cadence timing driven placer. The WNS and TNS are improved by 30.4% and 46.7% on average on the pre-set timing target, with 2% wire length increase (WL).

In next set of experiments, we show why the latch placement is important for timing. We fix the location of all sequential cells in the inial placements and

Pyramids -DP Experimental Results											
Circuits	Original (ps)			Pyramids (ps)			TAT	Change (%)			CPU
	WNS	*FOM*	*WL*	*WNS*	*FOM*	*WL*	(ps)	*WNS*	*FOM*	*WL*	(*s*)
s838_1	-117	-5415	29.3	-56	-1117	29.4	2100	52.3	79.4	0.1	2.5
s1196	-206	-16691	42.0	-189	-13733	42.4	1856	8.2	17.7	1.0	1.6
s1238	-110	-5388	53.3	-28	-981	53.9	1975	74.8	81.8	1.2	3.6
s1423	-232	-44414	44.6	-96	-7481	45.3	4168	58.5	83.2	1.7	6.2
s1488	-110	-3732	58.7	-91	-2999	58.7	1985	17.8	19.6	-0.0	3.5
s1494	-118	-2526	61.0	-17	-162	61.1	2122	85.9	93.6	0.2	3.2
s15850	-252	-27234	43.2	-219	-21862	43.1	4532	13.0	19.7	-0.3	6.2
s9234_1	-287	-47904	79.4	-244	-31104	79.8	5174	15.2	35.1	0.5	9.3
s13207	-398	-154110	89.7	-370	-112690	90.6	7165	7.1	26.9	0.9	33.0
s5378	-291	-60104	121.7	-205	-27543	122.6	5234	29.4	54.2	0.7	11.8
s38584	-528	-30212	807.2	-473	-22912	809.2	25857	10.4	24.2	0.2	107.7
s35932	-678	-299320	947.4	-620	-164599	950.5	33207	8.5	45.0	0.3	137.7
s38417	-975	-355126	782.0	-834	-258295	783.1	33130	14.5	27.3	0.1	76.9
avg								**30.4**%	**46.7**%	0.5%	

Table 5.2: Pyramids-DP Results on ISCAS benchmarks

run Pyramids-DP on combinational gate only. The results are report in Table 5.3. Therefore, Table 5.3 report how much timing improvement the Pyramids-DP is able to achieve by moving combinational cells only. Clearly, without moving the latches, the amount of timing improvement achievable is much less.

5.5.2 Pyramids-CP experiments in an industrial flow

In the following, we show the experimental results for Pyramids-CP within an industrial physical synthesis flow. First, we show how well our model resembles the realistic timing. Table 5.4 compares the model slack predicted to values measured by a commercial static timing analyzer. The latches are moved and nets are buffered before the measurement. Columns 2-4 report the initial, final, and improvement in worst slack, with the model presented in Section 5.4. The timing measured by STA engine is reported in Columns 5-7. Our observation is that there is a 97% correlation for the actual improvement compared with the model prediction, which implies the model is reasonable enough for optimization.

Results with Fixed Sequential Cells				
Circuits	Change (%)			CPU
	WNS	FOM	WL	(s)
s838_1	17.3	35.9	-0.4	4.6
s1196	10.5	17.6	1.0	1.7
s1238	44.8	64.5	0.4	4.0
s1423	32.1	46.5	-0.0	10.7
s1488	24.4	44.1	0.6	5.5
s1494	28.7	56.1	0.5	3.3
s15850	8.2	20.3	-1.4	9.4
s9234_1	9.6	29.8	4.1	17.4
s13207	2.2	6.6	-0.5	26.3
s5378	11.1	20.4	1.2	11.5
s38584	4.3	4.7	-0.1	51.0
s35932	7.4	21.2	0.1	90.7
s38417	14.2	29.7	0.2	67.5
avg	**16.5**%	**30.6**%	0.4%	

Table 5.3: Pyramids-DP results with fixed sequential cells

We use Pyramids-CP to improve the latch placement of an already optimized 130nm commercial ASIC with clock period 2.2ns and 3 million objects. We select the most critical latches that are not optimized, and we use the algorithm from [4] to perform buffering afterwards. We compare Pyramids-CP with the Center Of Gravity (COG) method, which is the current and intuitive solution in an industrial physical synthesis flow to improve the latch placements. In COG, the latch is placed to the center of gravity of adjacent pins. The "center" is weighted by the slack of all connected pins. The COG is tested in the same framework as the Pyramids. Table 5.5.2 shows a comparison between Pyramids and slack-weighted COG on 10 latches, which are the same latches are reported in Table 5.4. On average, the Pyramids-CP improves slack more than 40%, while the COG improves about 16%.

Model timing vs. reference timing						
	Model slack (ps)			Subcircuit slack (ps)		
Subcircuit	orig	new	imprv.	orig	new	imprv.
latch A0	-1147	-527	620	-953	-390	563
latch A1	-1090	180	1269	-897	356	1253
latch A2	-958	-36	923	-706	123	828
latch A3	-920	320	1241	-690	395	1085
latch A4	-920	312	1232	-690	173	863
latch A5	-964	296	1260	-681	376	1058
latch A6	-924	405	1328	-679	351	1030
latch A7	-913	213	1126	-633	290	923
latch A8	-876	342	1218	-614	440	1054
latch A9	-800	397	1198	-610	262	872
avg	-951	190	**1142**	-715	238	**953**

Table 5.4: The model used in Pyramids-CP is coherent with the actual timing model

5.6 Summary

We present Pyramids, an effective and efficient timing-driven placement algorithm. We propose two Pyramids formulations for different wire models, which are suitable for different stages in a physical synthesis flow. Pyramids-DP is based on the bounding box net model and is suitable for timing driven detailed placement. Pyramids-CP is based on the linear delay model, and is designed for critical paths refinement in later stage of a physical synthesis flow. Experimental results validate the effectiveness of both formulations. Pyramids-DP improved the timing of a set of circuits, which have already been optimized by Cadence SOC encounter, by 30.4% on average. The Pyramids algorithm-CP improved the slack by 40% of cycle time on average for a large commercial ASIC design.

Pyramids vs. Center-of-gravity						
	Pyramids Slack (ps)			COG Slack (ps)		
Subcircuit	orig	new	imprv.	orig	new	imprv.
latch A0	-953	-390	563	-953	-615	338
latch A1	-897	356	1253	-897	-78	819
latch A2	-706	123	828	-706	79	784
latch A3	-690	395	1085	-690	-690	0
latch A4	-690	173	863	-690	-690	0
latch A5	-681	376	1058	-681	-681	0
latch A6	-679	351	1030	-679	209	888
latch A7	-633	290	923	-633	-633	0
latch A8	-614	440	1054	-614	-614	0
latch A9	-610	262	872	-610	67	677
avg	-715	238	**953**	-715	-365	**351**

Table 5.5: Comparison of Pyramids-CP with the COG

Chapter 6

Total Power Optimization Combining Placement, Sizing and Multi-Vt Through Slack Distribution Management

6.1 Introduction

Beside timing closure, for nanometer IC designs (90nm and below), power dissipation has become one of the most important limiting factors since leakage is increasing exponentially as CMOS technology scales down. Both process and design technologies are being developed to conquer the leakage barriers. Among various design techniques, multiple-Vt assignment is very popular and effective. The idea is fairly straightforward. A design starts with all regular-Vt (R_{Vt}) cells. Once the timing target is roughly met, one replaces non-critical cells with their high-Vt (H_{vt}) counter parts, as the sub-threshold leakage current of a gate is exponentially related to the threshold voltage. Meanwhile, one need to fix the remaining failing paths by using a small number of low-Vt (L_{vt}) cells since they are faster (but leak much more).

The effectiveness of Vt swapping relies on the slack distribution. The slack distribution is heavily related with how timing closure is done during physical synthesis, e.g., placement and gate sizing. As timing and power are often conflicting objectives during optimization, traditionally, placement is mainly used for timing

optimization. And there is no existing work in placement that considers the leakage power reduction.

Gate sizing is used for both timing optimization and power reduction. Conventional gate sizing formulations either minimize the worst case delay or minimize the power under delay constraints [35, 10, 34, 20, 22, 28, 78]. However, gate sizing is never considered to help the Vt-swapping algorithm to maximize the power reduction overall, although Vt-swapping is known to be much more effective on leakage reduction.

To maximize the power reduction, the power saving opportunity in above physical design stages should be considered and utilized in a systematic manner. As we know the leakage current is exponentially related to the threshold voltage (Table 6.1), but linear to the cell size, multi-Vt assignment shall be a much more effective technique for leakage power reduction than gate sizing (i.e. by using high-Vt cells as much as possible). In other words, to reduce total power where leakage becomes prominent, it is more effective to use placement and gate sizing to *promote more effective Vt swapping afterwards* than using them independently for local power reduction. For example, we may size up some cells, which leads to less L_{vt} cells used finally. In that case, the amount of leakage saved could be much more than the power increased due to cells upsized.

In this chapter, we propose to use the slack distribution management to "glue" placement and gate sizing algorithms together to *boost* the Vt-swapping technique. The primary objective of our approach is to increase the sum of slacks on critical and near critical paths, i.e. to push the slack distribution curve (not the

Table 6.1: Normalized delay and leakage current for a cell with different threshold voltages in 65nm technology

Cell	L_{vt}	R_{vt}	H_{vt}
Delay	1	1.1	1.3
Leakage current	17.3	2.4	1

worst slack) of the circuit away from critical, even at the cost of up-sizing some cells slightly. Less total number of critical cells implies less L_{vt} cells and higher percentage of H_{vt} cells being used eventually. In other words, we trade small dynamic power increase for large leakage power reduction. In addition, we reduce the power directly by sizing down cells on non-critical paths when possible. Our methodology formulates a linear-programming (LP) based placement and two geometric programming (GP) based gate sizing algorithms to change the slack distribution.

In the rest of the chapter, section 6.2 motivates our proposed approach. The LP based placement stage is introduced in section 6.3. The GP formulations are in 6.4 and the Vt swapping algorithm is described in section 6.5. Experimental results are shown in section 6.6, and we summarize in section 6.7.

6.2 Motivation & proposed approach

In a typical flow, a design starts with all regular Vt cells (R_{vt}). A few timing violating paths that are very difficult to optimize in other ways can be fixed by swapping in L_{vt} cells. All R_{vt} cells on non-critical paths with large slack will be swapped into H_{vt} cells to save power. As shown in Table 6.1, the leakage of a H_{vt} cell is significantly smaller, about 17 times compared with a L_{vt} version at 65-nm

technology.

The results of Vt swapping is highly dependent on the slack distributions. If we can reduce the number of near critical cells, we may use fewer L_{vt} cells and more H_{vt} cells. Figure 6.1 plots the cell slack histogram of a circuit before and after placement plus gate sizing. The circuit is R_{vt} based. The slack histogram after optimization is tightened around a specified mean with a reduced deviation. Less near-critical cells implies less L_{vt} cells be used later, and less leakage power subsequently.

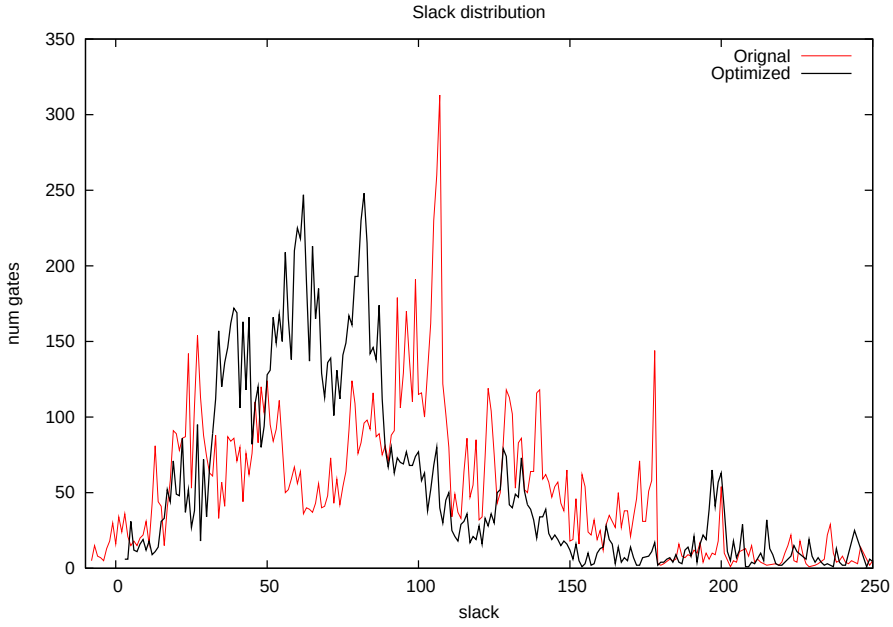


Figure 6.1: Slack distribution before and after optimization

6.2.1 The proposed flow

Our strategy is to use the placement and gate sizing to optimize the slack distribution to promote Vt swapping. We formulate a LP program for placement and a GP program for gate sizing to maximize the sum of slacks on semi-critical paths. In addition, cells on non-critical paths may be oversized even if they are swapped to H_{vt}. We formulate a GP problem to reduce slack and power on non-critical cells directly.

Algorithm 7 The Overall Algorithm

```
1:  The slack distribution management algorithm
2:      Input: initial design (all R_vt cells)
3:      while ( less than max. iter. & improved) do
4:          Incremental placement optimization
5:          TimingAnalysis
6:          Cell sizing on critical path for slack
7:          TimingAnalysis
8:          Size down non-critical cells
9:          TimingAnalysis
10: The Vt-swapping algorithm (Algorithm 2)
11: Functon: TimingAnalysis
12:     Pre-routing, and timing analysis
13:     if(improved) accept solution, annotate the database
```

We use placement and gate sizing iteratively to improve the slack distribution. Algorithm 7 shows our proposed flow. Starting from a design, we do the placement and critical cell gate sizing iteratively until no further improvement. Finally, we employee the Vt swapping to use a few L_{vt} cells to fix the remaining critical paths, and replace as many R_{vt} with H_{vt} cells as possible. At the end of each stage in the flow, we run a fairly accurate timing analyzer. The timing tool pre-routes the

circuit, extract the parasitics, and run the PrimeTime based timing analysis. The timing change from the previous stage is accurately updated and annotated back into the design databases, as the basis of the next stage.

6.2.2 Practical design constraints

In existing literature of power optimization, important practical design constraints, such as slew, noise, and short-circuit power, are often not considered, which makes the optimization algorithm impractical for realistic designs. For example, short circuit power is usually assumed small and ignored in most of existing power reduction algorithms. However, short circuit power may rise significantly if not explicitly controlled in the optimization framework.

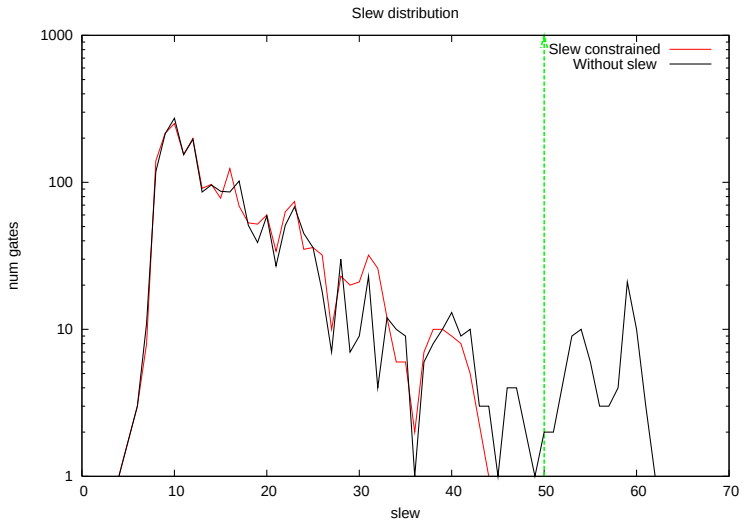


Figure 6.2: Slew rate distribution with and without explicit control

6.2.2.1 The slew and noise related constraints

Without restricting the maximum slew rate, cells on short paths will be over-downsized. Figure 6.2 plots the slew rate histogram of the gate sizing results with and without restricting the slew rate. in Figure 6.2, a lot of instances violate the 50 pico-second slew limit if ignoring slew constraints. Our maximum slew rate constraints set an upper bound for the slew rate. Furthermore, cells have different sensitivities to slew for noise. Our model includes the effective fan-out constraints, which is an effective way to reduce the noise related issues.

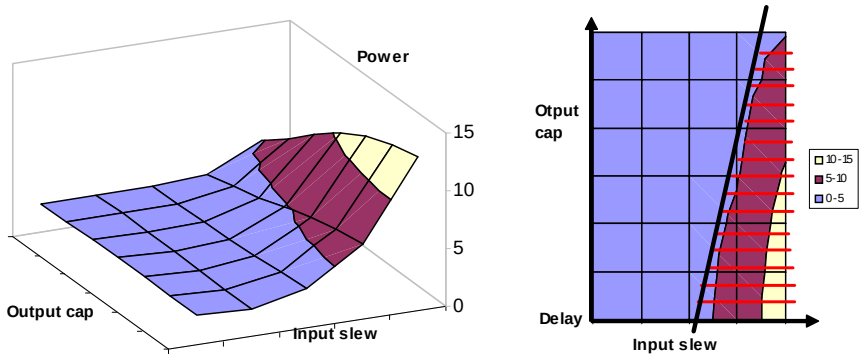


Figure 6.3: A simple yet effective short circuit power constraint model.

6.2.2.2 Short circuit power constraint

Short circuit power is difficult to model and ignored in most existing power optimization algorithms. The inputs for internal power are the input slew and output capacitance, as shown in Figure 6.3. Imaging a scenario that the input slew of a cell is large and the load can be charged full quickly, the PMOS and NMOS will be both on for a longer period. The Vdd to ground current will consume a lot of power.

Short circuit power is often assumed very small. However, in our experiments, it is comparable to leakage if not properly handled. Figure 6.3 is a SPICE simulation based look-up table to interpolate the short circuit power. Note that short circuit power could rise dramatically if the ratio of the input slew and output capacitive load falls into a certain range, e.g. input slew is large while the cell is driving a comparably small load.

In later sections, we will show how to handle above important design constraints in our proposed algorithm.

6.3 LP based placement for power

The objective of our LP placement formulation is to reduce the power in Vt-swapping stage incrementally. Therefore, instead of reducing the worst case delay, our LP based placement is formulated primarily to reduce the total number of critical and semi-critical cells, i.e. to push the slack curve away from the critical point, which helps the Vt-swapping tool on leakage reduction.

Linear programming is commonly used for incremental timing driven placement [22, 39, 25, 26, 58]. In LP based incremental placement, a few critical paths are selected by a sign off timer, and critical paths are optimized incrementally. Existing LP based timing driven placement algorithms use the half parameter wire length for wire estimation, as HPWL can be formulated exactly in a LP framework.

Chen et.al. [22] proposed a simultaneous placement and gate sizing approach to optimize the delay. Because the unified placement and sizing GP formu-

lation is not convex, the problem was formulated into a generic geometric program (GGP) and solve iteratively. However, the HPWL based wire load estimation is much less accurate compared with that in a stand alone gate sizing problem, which can be measured separately. A simultaneous formulation will make the wire load estimation less accurate for gate sizing, which often results in a sub-optimal solution.

6.3.1 The LP formulations

We assume the following gate delay DP_i and transition SP_i models for cell i

$$DP_i = dp_I + a1_i \cdot Slew_i + a2_i \cdot Cap_i \tag{6.1}$$

$$SP_i = sp_I + u1_i \cdot Slew_i + u2_i \cdot Cap_i \tag{6.2}$$

where $a1_i$, $a2_i$, $u1_i$, and $u2_i$ are the fitting coefficients. dp_I and sp_I denote the intrinsic delay and slew of the corresponding pin of the cell. $Slew_i$ denotes the input slew. Let $HPWL_j$ denotes the HPWL of net j, and Cap_j represents the capacitive load of the driver of net j. We have

$$Cap_j = c \cdot HPWL_j + Cpin_j$$

which is the sum of the wire capacitance $cHPWL_j$ plus the total pin capacitance driven by net j, and c is the unit capacitance.

The LP placement algorithm selects a few critical paths selected from the timing report, which have slacks less than a threshold. The net delay sensitivity is computed for each critical net, and a LP program is formulated to minimize the

sum of the weighted critical nets, which is an indirect method to increase the sum of total slack on those critical paths. The net delay sensitivity Sn_j is based on the delay propagation sensitivity computation in [58]

$$Sn_j = c \cdot (a2_i + a1_{i+1} \cdot u2_i)$$

Elmore delay [33] is used for wire delay modeling and the symbols related with net delay are omitted in the formulation for simplicity.

Similar to [56], the critical paths were counted to compute the criticality of each selected critical net, the criticality score of net j is denoted by Sc_j. Therefore, the combined timing weight $wt_t = Sc_j Sn_j$. The dynamic power is a function of the load capacitance of the net. If cell i drives net j, we have a power weight

$$wt_p = 0.5\alpha_i \cdot F \cdot V^2 \tag{6.3}$$

where α_i denotes the switching rate, F is the frequency, and V is the voltage. A control parameter β is used to adjust the ratio between the timing and power weight.

$$wt_j = \beta wt_p + (1-\beta) wt_t$$

β is a value between 0 and 1. The primary objective of our LP placement is to reduce the leakage power, thus, β is set to a relatively small value. A LP program is formulated to minimize the sum of the weighted critical nets, which indirectly increases the sum of pin slacks.

$$min \ \ \sum wt_j L_j$$

$$\forall j \in Selected \ \ critical \ \ nets$$

The residual overlap created in this stage is carefully removed.

6.4 GP based gate sizing for power

Placement has a limited impacton slack distribution improvement if the cell sizes are not changed. To push the slack curve further, we use the effort based delay model, and formulate a *Geometric programming* based gate sizing problem. GP is a special type of the non-linear optimization problem that has been used for gate sizing since the 80s [27, 11, 12]. The standard GP problem has a posynomial objective and special format constraints. In last ten years, the solving efficiency of GP is approaching that of *Linear Programming*. We refer the reader to a tutorial for geometric programming [12].

Conventional gate sizing formulations minimize the worst case delay in a circuit with power or area constraints [10, 12, 22], or minimize the power directly under the delay constraints [78]. On the contrary, our first GP formulation increases the sum of slack on critical and near critical outputs instead. Our second GP program is related to the conventional formulation, which focuses on the non critical part of the circuit to "absorb" large slacks. Therefore, we treat cells differently depending on the criticality of the cell.

6.4.1 Cell classification

Cells are classified into two sets, the non-critical set *NC* and the critical set *CRIT*, based on the output pin slack. If the pin slack is larger than a threshold, we add the cell into *NC*. Similarly, if the slack is small enough, we add the cell into *CRIT*.

For the first GP program, we start from all outputs with slack less than δ,

for example, $\delta = 70$ ps. We traverse the circuit in a reversed breath first order, and the reversed BFS traversal proceeds only on cell outputs with the slack smaller than $\delta + \gamma$ and stops at signal inputs, which are the inputs of the circuit or the outputs of sequential cells. Only cells with slack less than θ ($\theta < \delta$) are selected into the *CRIT*. The size of cells in *CRIT* are variables for the GP program. As the arrival time of all outputs with slack less than δ is controlled in the GP program, $\delta - \theta$ acts as a guard band to ensure that timing on other outputs are not disturbed too much. For the second GP program for non-critical cells, all cells with a slack larger than a threshold are sizable cells in *NC*, and all outputs are included into the GP problem. In other words, all arrival times are controlled.

6.4.2 The GP models

We model the gate as a resistor and a switch that drives a RC network. The gate delay and transition are the functions of the gate size W and the total capacitive load, Cap. The equation for the cell equivalent impedance is different for delay and transition equations, and the delay models for each pin of a gate and that for the falling or rising transition are different. We use the worst case models for a cell. The gate delay Dg_i and slew Sg_i are given by

$$Dg_i = dg_I + (h_i/W_i) \cdot Cap_i \tag{6.4}$$

Slew is not propagated. But slew is monitored and restricted by the following equation

$$Sg_i = sg_I + (v_i/W_i) \cdot Cap_i \tag{6.5}$$

Cap is the sum of the capacitive load and the gate capacitance a cell drives. The pin capacitance of a cell i is a linear function of the cell size W_i.

$$Cp_i = e_i + f_i W_i \tag{6.6}$$

In above equations, dg_I, h_i, sg_I, v_i, e_i, and f_i are all fitting coefficients to the cell library.

Assuming a cell i drives a sizable cells and b non-sizable cells. The total capacitance the cell i drives is

$$Cap_i = \sum_{k=1}^{a} (e_k + f_k W_k) + \sum_{l=1}^{b} (Cp_l) + Cap_{wire} \tag{6.7}$$

We add the wire delay in our formulation. An accurate pre-routing tool is used to estimate routs. The pin to pin wire delay is computed by a static timer and treated as a constant in the gate sizing formulation.

Three major source of power consumption, including dynamic, short circuit, and leakage power are considered in our approach. The dynamic power can be written as

$$P_i = 0.5\alpha \cdot F \cdot V^2 \cdot Cap_i \tag{6.8}$$

where α, F, and V are defined in Equation (6.3). The leakage power is assumed proportional to the gate size, and the parameter $leak$ is extracted from the SPICE simulation based power library. The following linear leakage model is sufficient for the leakage estimation in the gate sizing stage.

$$L_i = leak_i \cdot W_i \tag{6.9}$$

The short circuit power is modeled as constraints in the GP formulation.

6.4.3 Gate sizing effectiveness analysis

Slack and power optimization are often contradictory objectives. To reduce the delay by sizing up cells will increase the dynamic and the leakage power. Whether or not and how to size a cell should be also determined by if such a chance has negative overall effect potentially. We do the following gate sizing effectiveness analysis to estimate a sizing range, i.e. we do not size a cell exceeding a limit that may have a negative effect. In the following, we will derive the power and delay

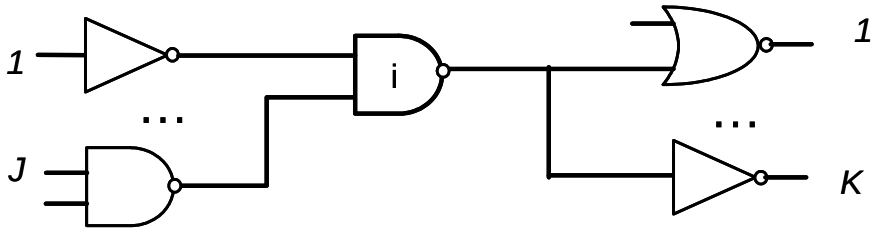


Figure 6.4: Gate sizing effectiveness analysis

sensitivity to cell size. In Figure 6.4, the cell i has J inputs and drives K downstream cells. If we change cell i from R_{vt} to L_{vt}, the associated power will change by ΔPv_i, and delay will change by ΔDv_i. We have

$$\Delta Pv_i = \Delta Leak_i + \alpha_i FV^2 \sum_j f_j \Delta Cp_i, j = 1..J$$

$$\Delta Dv_i = ((h_i^{'} - h_i)/W_i)(\sum_k Cp_k + Cw) + max(\frac{h_j}{W_j}\Delta Cp_i), k = 1..K$$

Where ΔCp_i is the pin capacitance change. $h_i^{'}$ is the corresponding coefficient for L_{vt} cell, as in Equation (6.4). Similarly, if changing the cell size by ΔW_i, the associated power change is denoted by ΔPg_i,

$$\Delta Pg_i = Leak_i \Delta W_i + \alpha_i FV^2 \sum_j h_j f_j \Delta W_i$$

The associated delay change is denoted by Dg_i

$$\Delta Dg_i = (\frac{h_i}{W_i + \Delta W_i} - \frac{h_i}{W_i}) \sum_k (Cp_k + Cw) + max(\frac{h_j}{W_j} \Delta W_i)$$

To solve the equation $\Delta Pv_i / \Delta Dv_i = \Delta Pg_i / \Delta Dg_i$, one zero and one non-zero ΔW_i solutions are generated. If the non-zero solution is negative, sizing up cell i will increase both power and delay, and the cell i is not allowed to be sized up. If one of the solutions is positive, and $W_i + \Delta W_i < Wmax_i$, we set the maximum sizable range of cell i as $Wmax_i = W_i + \Delta W_i$. Beyond this limit, gate sizing has a lower power and delay benefit compared to Vt swapping.

6.4.4 GP for near-critical cells

In brief, our first GP program creates more slacks for near critical cells, which maximizes the sum of slacks on critical outputs. The GP formulation is given by

$$\begin{aligned} minimize & \sum_j AT_j \\ & + \sum_i wt_i W_i (\Delta Pg_i / \Delta W_i), j \in PO \cap CRIT, i \in CRIT \qquad (6.10) \\ s.t.\ & Dg_i = d_I + \frac{b_i}{W_i} Cp_i \\ & AT_i \geq Dg_i + max(AT_{i-1}^p), p \in input\ pins\ of\ i \\ & AT_i = T_{start}, \forall i \in PI \\ & W_i \geq Wmin_i, W_i \leq Wmax_i \end{aligned}$$

In above, AT_i is the arrival time at the output of the $cell_i$. $\Delta Pg_i / \Delta W_i$ is the

power sensitivity to cell sizes

$$\Delta Pg_i/\Delta W_i = Leak_i + \alpha_i FV^2 \sum_j h_j f_j \tag{6.11}$$

T_{slack} is a slack threshold. In the above GP formulation, we optimize the sum of the arrival time of all critical and near critical outputs. $Wmax_i$ and $Wmin_i$ are the sizing range for cell i. Wire delay is not shown for clarity, which is a constant computed by a static timer conjuncted with a pre-routing tool.

Let wt_i denotes the power weight. Without the power objective $\sum_i wt_i W_i$, the cell could be overly unsized, which will cause unnecessary increase on power. Before the optimizations, the sum of the arrival time on critical outputs and the sum of dynamic and leakage power on cells in $CRIT$ are evaluated. The power weight is computed to normalize the arrival time and power objectives, and the power weight is set to be associated with the power sensitivity of each cell. A 0 to 1 parameter is set to adjust the ratio between the arrival time and power objects.

6.4.5 GP for non-critical cells

The GP for non-critical cells is to optimize the total power on high slack cells, such that the arrival time does not violate timing constraints. The GP problem for non-critical cells can be written as

$$\begin{aligned} &min. \sum_i (\Delta Pg_i/\Delta W_i), i \in NC \\ &s.t.\ AT_i \leq max((T_{cycle} - T_{threshold}), AT\_orig_i), i \in PO \end{aligned} \tag{6.12}$$

where $\Delta Pg_i/\Delta W_i$ is from Equation (6.11). $T_{threshold}$ is the slack guard band. We consider swapping non-critical cells with slack larger than $T_{threshold}$ to H_{vt} cells.

AT_orig_i is the original arrival time of the output i. Constraint (6.12) implies that for each output i, the arrival time after the optimization may not violate the larger of a delay threshold and its original delay. The shared constraints in above GP problems are not shown here for simplicity.

6.4.6 Modeling important constraints

Besides the delay and power, there are a few constraints that are critical for industry practices, for example, the maximum slew constraint, the effective fan-out constraint for noise, and the short circuit power constraints, which were often ignored in previous studies. Our formulation considers those constraints and model them as follows in the GP framework.

6.4.6.1 The max slew constraint

Although adding the slew constraints will significantly limit the amount of power reducible, we should not ignore the slew constraints because slew rate violations are unacceptable for real world designs. The slew Equation (6.5) is used to estimate the slew rate, and we use the following to transform the slew constraint into sizing constraint in GP form.

$$\begin{aligned} S_i &= s_I + \frac{v_i}{W_i} Cp_i \\ S_i &\leq Slew_{max} \end{aligned} \tag{6.13}$$

where $Slew_{max}$ is the maximum slew rate acceptable.

6.4.6.2 Effective fan-out constraint for noise tolerance

The concept of effective fan-out (Efo) is related to but different from the conventional fan-out concept. Efo is the ratio of the effective capacitance a cell drives divided by the effect impedance ratio of the driver compared to a standard inverter. The effective impedance $Ratio$ is the hold resistance of a cell divided by that of a standard inverter at a certain voltage level. The Efo constraint is given by

$$Efo_i = \frac{Cp_i}{Ratio_i \times C_{inv1}} \leq Efo_{limit} \tag{6.14}$$

Applying an effective fan-out constraint on each cell will avoid introducing large amount of noise issues during the optimization.

6.4.6.3 Short circuit power constraint

The short circuit power is non-trivial to handle, and mostly ignored in previous power optimization work. Since the short circuit power is not large unless the ratio of the input slew and output capacitive load falls into a certain range, as shown in Figure 6.3, we can specify a *do-not-enter* region by adding a linear constraint to restrict the ratio between the input slew and output capacitance to avoid large short circuit power consumption.

$$Cap_i \geq p_i + q_i S_i \tag{6.15}$$

where Cap_i is the capacitive load driven by cell i. p_i and q_i are the parameter of the linear function shown in Figure 6.3, which specify the boundary of the do-not-enter zone. Above constraint ensures that the input slew of a cell should not be much larger than its output slew.

The number of possible sizes for a gate varies depending on the gate type. An inverter could have over 20 different sizes. Our algorithm assumes the sizes are continuous. The solution of the GP solver are continuous gate sizes, which will be mapped into the closest discrete ones. The discrete size mapping stage may introduce less than 5 percent errors.

Algorithm 8 The Vt-swapping algorithm

```
1:  Input The design after placement and sizing opt.
2:  while (stopping criteria not meet) do
3:     foreach (all cells)
4:        if (slack > High) swap to H_vt
5:        if (slack < Low) swap to L_vt
6:     end
7:     TimingAnalysis
8:     Sort cells on Senstivity (critical and noncritical list)
9:     foreach (Sorted cells)
10:       Swap to L_vt or R_vt
11:       Propagate timing and evaluate
12:    end
13: end
```

6.5 Vt swapping algorithm

We use a multiple pass sensitivity based Vt swapping algorithm, as shown in Algorithm 8 to swap cells. Cells with very large or small slacks are processed first. The rest are sorted on their sensitivity score. In each swapping pass, two hashes are created, one for R_{vt} cells and the other for L_{vt} cells. The sensitivity of a cell is computed by the original slack of the cell, the up-cone impact and the down cone impact of the cell. One top cell is selected at a time. The internal timer

Table 6.2: Total power comparison

	65 nm		Total power (mw)				Improvement %		
	Gates	Nets	Base	VT	PV	PGV	$VT\|B$	$PV\|B$	$PGV\|B$
ckt1	1765	2360	29.79	24.60	22.06	20.52	17.4	25.9	31.1
ckt2	2334	2881	30.26	22.41	21.93	19.69	25.9	27.5	34.9
ckt3	6640	8644	142.51	103.45	101.31	96.11	27.4	28.9	32.6
ckt4	9254	7928	110.86	93.57	93.57	86.38	15.6	15.6	22.1
ckt5	9541	9539	233.56	151.81	147.23	123.40	35.0	37.0	47.2
ckt6	12716	14042	241.27	155.89	154.14	140.43	35.4	36.1	41.8
ckt7	15486	18360	287.22	233.63	226.96	217.14	18.7	21.0	24.4
ckt8	27103	26991	499.15	377.34	372.51	354.58	24.4	25.4	29.0
							25.0	27.2	32.9

propagates the timing changes down stream and upstream to update the required times and the slacks. The process continues until the slack requirement is met. The swapping process will be performed multiple times for different supply voltages and performance corners. A solution that satisfies all corners will be adopted.

6.6 Experimental Results

The placement and gate sizing algorithm are implemented in C++ and the Vt swapping algorithm is in perl. We use the commercial tool MOSEK [64] as the GP solver. Several modules from a multi-GHz micro-processor in 65nm process technology are used for experiments. The number of cells and nets are shown in table 6.2, which are typical in micro-processor designs. The circuits have been initially manually placed and timing optimized and taped out in a test chip. It is to be noted that the high performance microprocessor circuits have a stringent timing target and are very difficult for timing optimization. Therefore, the multi-Vt swapping technique has to be used to repair the remaining failing paths, in most of

cases. All experiments are tested on a 2.4GHz 64-bit Opteron Linux server. We use an internal power evaluation tool to estimate the power consumption.

Table 6.2 shows the total power comparisons. Table 6.3 and 6.4 report the comparisons of leakage power and dynamic power respectively. In all tables, column *Base* shows the base-line optimization condition where cells are mostly R_{vt} cells. Column VT shows the power after the Vt swapping, and *BASE* stands for the baseline. PV shows the combined placement and Vt swapping, and column *PGV* stands for the combined placement, gate sizing and Vt swapping flow. We can see that the Vt swapping is very effective in reducing leakage power. The combined LP based placement and GP based gate sizing algorithm provides additional improvement and the flexibility to trade off on dynamic and static power through optimizing the slack distribution. We observe an additional 7.9% total power reduction, which is significant for manually optimized custom circuits. In current configurations, the placement optimization is configured to mostly help leakage power. The combined placement, gate sizing and Vt swapping gives the best results and helps to reduce 63.8% of leakage power and 32.9% of total power consumption.

The break down of runtime is shown in table 6.5. Column *Timing* reports the runtime of the static timing analysis flow. Our sophisticated timing analysis flow pre-routes the circuit, extracts parasitics and run a PrimeTime engine to generate the timing report and annotates the timing information into the design database. We run the timing analysis at the end of every optimization stage to update the timing information. Therefore, multiple runs of the timing analysis flow took a lot of runtime. The break down of dynamic and leakage power in circuits before and after

Table 6.3: Leakage power comparison

	Base	VT	PGV	$VT\|Base$ %	$PGV\|Base$ %
ckt1	10.50	6.09	3.28	42.0	68.8
ckt2	11.49	4.79	3.67	58.3	68.1
ckt3	52.11	20.10	17.38	61.4	66.6
ckt4	45.76	30.42	28.06	33.5	38.7
ckt5	93.04	26.62	18.28	71.4	80.4
ckt6	99.29	24.78	19.64	75.0	80.2
ckt7	104.77	60.25	49.86	42.5	52.4
ckt8	215.24	108.46	96.28	49.6	55.3
				54.2	63.8

Table 6.4: Dynamic power comparison

	Base	VT	PGV	$VT\|Base$ %	$PGV\|Base$%
ckt1	19.29	18.51	17.24	4.0	10.6
ckt2	18.77	17.62	16.02	6.1	14.7
ckt3	90.40	83.35	78.73	7.8	12.9
ckt4	65.10	63.15	58.32	3.0	10.4
ckt5	140.52	125.19	105.12	10.9	25.2
ckt6	141.98	131.11	120.79	7.7	14.9
ckt7	182.45	173.38	167.28	5.0	8.3
ckt8	283.91	268.88	258.30	5.3	9.0
				6.2	13.3

optimization is reported in Figure 6.5. We observe significant leakage reduction after applying our power optimization algorithm. The leakage reduction is due to higher percentage of H_{vt} cell used after the power optimization. The percentage of different Vt cells in all circuits are illustrated in figure 6.6. Originally a few circuits have negative slacks, and all circuits meet the timing target after the optimization. More L_{vt} cells is used in circuit 4 to close timing, and the percentage of L_{vt} cells is relatively small for the rest of testcases. Therefore, leakage power is reduced

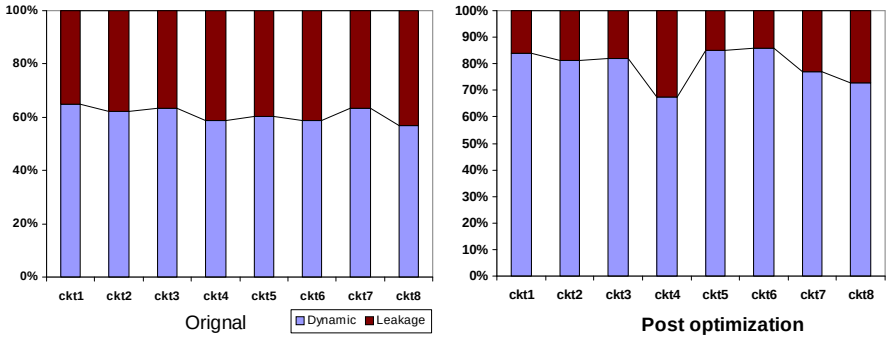


Figure 6.5: The leakage and dynamic power break up before and after optimizations significantly.

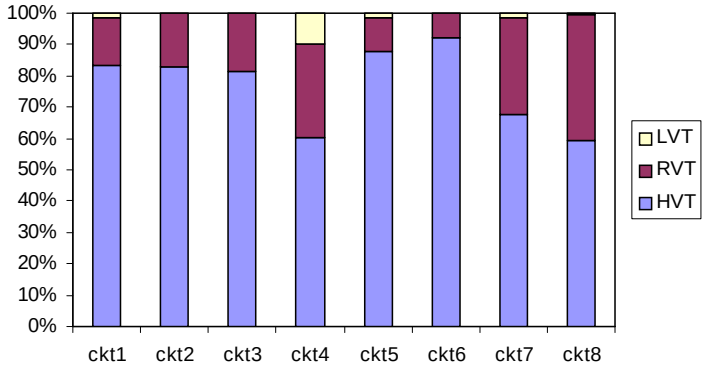


Figure 6.6: The percentage of different threshold voltage cells

6.7 Summary

In this chapter, we propose to combine placement, gate sizing, and multiple-Vt cell swapping algorithm and methodology for leakage and total power optimization. Our unified slack distribution management strategy makes it possible to com-

Table 6.5: Runtime breakup(s)

Testcases	*Place*	*Size*	*Swap*	*Timing*
ckt1	4	14	55	168
ckt2	3	12	69	90
ckt3	30	75	124	453
ckt4	25	42	137	217
ckt5	26	68	149	324
ckt6	32	228	171	262
ckt7	16	193	186	342
ckt8	43	384	245	538

bine different technique, the placement and gate sizing, to maximize the power reduction, while satisfying timing constraints. Our placement and gate sizing problems are formulated to optimize the slack distribution, which in turn transformed into power reduction through Vt-swapping. Our approach treats cells differently depending on their timing criticality. On a set of timing-closed multi-gHz 65nm custom microprocessor circuits, our approach reduced the leakage power by 63.8% and the overall power by 32.9%. Various practical design constraints are incorporated in our approach which were not considered before. Since power is becoming one of the most important design objectives, we believe there is a lot of room for future research on the total power reduction.

Chapter 7

Conclusions

Major optimization objectives in physical synthesis include the wire length, timing, power, etc. In this dissertation, we present several placement driven optimization algorithms to advance the state of art of multi-objectives VLSI design closure.

The first work is DPlace, an analytical placement engine that scales well to large scale circuit placement. DPlace employs a two-stages strategy that decouples the cell spreading and wire length minimization tasks and reduces the complexity of each placement iteration, which in turns brings the efficiency to DPlace. The flexibility in DPlace framework makes it possible to explicitly control the cell movement during the diffusion spreading stage in every DPlace iteration, which has the potential advantages for ECO and timing driven placement, where precise cell movement control is desired. Furthermore, The DPlace framework makes it flexible to integrate additional wire length and congestion optimization techniques.

The nature of design optimizations is often incremental. An incremental optimization tool normally starts from an existing placement solution and addresses a few design violations in small steps. The cell overlaps introduced between incremental optimization steps need to be smoothly resolved by placement migration. I

developed a novel computational geometry based placement migration framework to address placement migration problem. Our experimental results on legalization problem have demonstrated significant improvements on wire length and stability for placement migration.

A smooth placement migration only implicitly preserves timing. I proposed a LP based critical path optimization tool to explicitly improve the timing in high performance custom designs and ASICs. The new framework uses an accurate delay sensitivity based net-weighting method that combines the advantage of the path-based approach. To avoid large disturbance to existing image of the design, I introduced a novel criticality adjacency network concept to control the timing perturbation precisely during the optimization.

The Pyramids algorithm presented is an effective and efficient timing-driven placement algorithm, which has tow formulations, the Pyramids-DP and Pyramids-CP. Pyramids-DP is based on the bounding box net model and is suitable for timing driven detailed placement. Pyramids-CP is based on the linear delay model for timing arcs. Pyramids-CP is designed for critical paths refinement in later stage of a physical synthesis flow, where many buffers are inserted. The

To address the increasing concerns for power consumption, I propose to combine placement, gate sizing, and multiple-Vt cell swapping algorithm and methodology for leakage and total power optimization. We use the slack distribution management as the centralized objective for incremental placement, gate sizing and Vt-swapping, which leads to the maximized power reduction. The placement and gate sizing problems are formulated to optimize the slack distribution, which in turn

transformed into power reduction through Vt-swapping. In addition, various practical design constraints are incorporated in our approach, which were often ignored before. Furthermore, we believe that various optimization techniques, such as the gate sizing and buffering need to be incorporated with placement to speed up the process of multi-objective VLSI design closure.

Bibliography

[1] A. Agnihotri, M. C. Yildiz, A. Khatkhate, A. Mathur, S. Ono, and P. H. Madden. Fractional cut: improved recursive bisection placement. In *Proc. Int. Conf. on Computer Aided Design*, pages 307–310, 2003.

[2] Ameya R Agnihotri and Patrick H Madden. Fast analytic placement using minimum cost flow. In *Proc. Asia and South Pacific Design Automation Conf.*, 2006.

[3] Ameya R. Agnihotri, Satoshi Ono, and Patrick H. Madden. Recursive bisection placement: feng shui 5.0 implementation details. In *Proc. Int. Symp. on Physical Design*, 2005.

[4] C. J. Alpert and et al. Fast and flexible buffer trees that navigate the physical layout environment. In *Proc. Design Automation Conf.*, 2004.

[5] Charles J. Alpert, Chris Chu, and Paul G. Villarrubia. Physical synthesis comes of age. In *Proc. Int. Conf. on Computer Aided Design*, 2007.

[6] Charles J. Alpert and et. al. Techniques for fast physical synthesis. In *Proc. IEEE*, 2007.

[7] Charles J. Alpert and et.al. Accurate estimation of global buffer delay within a floorplan. In *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 2006.

[8] Charles J. Alpert, G.-J. Nam, P. Villarrubia, and M. C. Yildiz. Placement stability metrics. In *Proc. Asia and South Pacific Design Automation Conf.*, Jan, 2005.

[9] Semiconductor Industry Association. *International Technology Roadmap for Semiconductors*, 2005.

[10] M. Berkelaar and J. Jess. Gate sizing in MOS digital circuits with linear programming. In *Proc. European Design Automation Conf.*, pages 217–221, June 1990.

[11] Stephen P. Boyd and Seung Jean Kim. Geometric programming for circuit optimization. In *Proc. Int. Symp. on Physical Design*, 2005.

[12] Stephen P. Boyd, Seung Jean Kim, Lieven Vandenberghe, and Arash Hassibi. A totorial on geometric programming. *Technical Report, ISL, Electrical Engineering Department, Stanford University*, 2004.

[13] U. Brenner, A. Pauli, and J. Vygen. Almost optimum placement legalization by minimum cost flow and dynamic programming. In *Proc. Int. Symp. on Physical Design*, pages 2–9, 2004.

[14] U. Brenner and J. Vygen. Faster optimal single-row placement with fixed ordering. In *Proc. Design, Automation and Test in Eurpoe*, pages 117–121, 2000.

[15] Michael Burstein and Mary N. Youssef. Timing influenced layout design. In *Proc. Design Automation Conf.*, pages 124–130, 1985.

[16] Andrew E. Caldwell, Andrew B. Kahng, and Igor L.Markov. Can recursive bisection alone produce routable, placements? In *Proc. Design Automation Conf.*, pages 477–482, 2000.

[17] Tony Chan, Jason Cong, Joseph Shinnerl, Kenton Sze, and Min Xie. mpl6: enhanced multilevel mixed-size placement. In *Proc. Int. Symp. on Physical Design*, 2006.

[18] Tony Chan, Jason Cong, and Kenton Sze. Multilevel generalized force-directed method for circuit placement. In *Proc. Int. Symp. on Physical Design*, 2005.

[19] Chunhong Chen, Xiaojian Yang, and Majid Sarrafzadeh. Potential slack: an effective metric of combinational circuit performance. In *Proc. Int. Conf. on Computer Aided Design*, pages 198–201, 2000.

[20] G. Chen, H. Onodera, and K. Tamaru. An iterative gate sizing approach with accurate delay evaluation. In *Proc. Int. Conf. on Computer Aided Design*, pages 422–427, November 1995.

[21] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen, and Y.-W. Chang. A high quality analytical placer considering preplaced blocks and density constraint. In *Proc. Int. Conf. on Computer Aided Design*, 2006.

[22] W. Chen, C. Hseih, and M. Pedram. Simultaneous gate sizing and placement. In *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, pages 206–214, February 2000.

[23] Yongseok Cheon, Pei-Hsin Ho, Andrew B. Kahng, Sherief Reda, and Qinke Wang. Power-aware placement. In *Proc. Design Automation Conf.*, 2005.

[24] D. G. Chinnery and K. Keutzer. Closing the gap between asic and custom: an asic perspective. In *Proc. Design Automation Conf.*, pages 637–642, 2000.

[25] Wonjoon Choi and Kia Bazargan. Incremental placement for timing optimization. In *Proc. Int. Conf. on Computer Aided Design*, page 463, 2003.

[26] Amit Chowdhary, K. Rajagopal, S. Venkatesa, T. Cao, V. Tiourin, Y. Parasuram, and B. Halpin. How accurately can we model timing in a placement engine? In *Proc. Design Automation Conf.*, pages 801–806, 2005.

[27] C. Chu and D. Wong. Vlsi circuit performance optimization by geometric programming. In *Annals of Operations Research*, pages 105:37–60, 2001.

[28] W. Chuang and S. S. Sapatnekar. Power vs. delay in gate sizing: Conflicting objectives? In *Proc. Int. Conf. on Computer Aided Design*, pages 463–466, November 1995.

[29] Jason Cong, Lei He, Cheng-Kok Koh, and Patrick H. Madden. Performance optimization of VLSI interconnect layout. *Integration, the VLSI Journal*, 21:1–94, 1996.

[30] Jason Cong, Tim Kong, Joseph R. Shinnerl, Min Xie, and Xin Yuan. Large-scale circuit placement: Gap and promise. In *Proc. Int. Conf. on Computer Aided Design*, 2003.

[31] Wilm E. Donath, Reini J. Norman, Bhuwan K. Agrawal, Stephen E. Bello, Sang Yong Han, Jerome M. Kurtzberg, Paul Lowy, and Roger I. McMillan. Timing driven placement using complete path delays. In *Proc. Design Automation Conf.*, pages 84–89, 1990.

[32] H. Eisenmann and F. M. Johannes. Generic global placement and floorplanning. In *Proc. Design Automation Conf.*, pages 269–274, 1998.

[33] W. C. Elmore. The transient response of damped linear networks with particular regard to wide-band amplifiers. *Journal of Applied Physics*, 19(1):55–63, January 1948.

[34] S. S. Sapatnekar et.al. An exact solution to the transistor sizing problem for CMOS circuits using convex optimization. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 1993.

[35] J. P. Fishburn and A. E. Dunlop. TILOS: A posynomial programming approach to transistor sizing. In *Proc. Int. Conf. on Computer Aided Design*, pages 326–328, November 1985.

[36] Fortune. Voronoi diagrams and delaunay triangulations. In *Computing in Euclidean Geometry, 2nd Edition, World Scientific, Lecture Notes Series on Computing – Vol. 1*. 1992.

[37] Padmini Gopalakrishnan, Altan Odabasioglu, Lawrence Pileggi, and Salil Raje. Overcoming wireload model uncertainty during physical design. In *Proc. Int. Symp. on Physical Design*, pages 182–189, 2001.

[38] Bill Halpin, C. Y. Roger Chen, and Naresh Sehgal. A sensitivity based placer for standard cells. In *Proc. of Great Lakes symp. on VLSI*, pages 193–196, 2000.

[39] Bill Halpin, C. Y. Roger Chen, and Naresh Sehgal. Timing driven placement using physical net constraints. In *Proc. Design Automation Conf.*, pages 780–783, 2001.

[40] D Hill. Method and system for high speed detailed placement of cells within an integrated circuit design. US patent 6,370,673, 2002.

[41] Bo Hu and Malgorzata Marek-Sadowska. Far: fixed-points addition & relaxation based placement. In *Proc. Int. Symp. on Physical Design*, 2002.

[42] S. W. Hur and J. Lilis. Mongrel: hybrid techniques for standard cell placement. In *Proc. Int. Conf. on Computer Aided Design*, pages 165–170, 2000.

[43] ISPD_2002_Benchmark. http://vlsicad.eecs.umich.edu/bk/ispd02bench/.

[44] ISPD2002BenchmarkModified. http://gibbon.uwaterloo.ca/research.html. 2002.

[45] ISPD_2005_Placement_Contest. http://www.sigda.org/ispd2005/ispd05/slides/10-1-placement-contest-ispd05.ppt.

[46] Yun-Cheng Ju and Resve A. Saleh. Incremental techniques for the identification of statically sensitizable critical paths. In *Proc. Design Automation Conf.*, pages 541–546, 1991.

[47] A. B. Kahng, S. Reda, and Q. Wang. Aplace: A general analytic placement framework. In *Proc. Int. Symp. on Physical Design*, pages 233–235, April 2005.

[48] A. B. Kahng, S. Reda, and Q. Wang. Architecture and details of a high quality, large-scale analytical placer. In *Proc. Int. Conf. on Computer Aided Design*, November 2005.

[49] A. B. Kahng, P. Tucker, and A. Zelikovsky. Optimization of linear placements for wirelength minimization with free sites. In *Proc. Asia and South Pacific Design Automation Conf.*, pages 18–21, 1999.

[50] A. B. Kahng and Q. Wang. An analytic placer for mixed-size placement and timing-driven placement. In *Proc. Int. Conf. on Computer Aided Design*, pages 565–572, November 2004.

[51] Andrew B. Kahng, Stefanus Mantik, and Igor L. Markov. Min-max placement for large-scale timing optimization. In *Proc. Int. Symp. on Physical Design*, pages 143–148, 2002.

[52] Andrew B. Kahng, Igor L. Markov, and Sherief Reda. On legalization of row-based placements. In *ACM Great Lakes Symposium on VLSI*, pages 214–219, 2004.

[53] Andrew B. Kahng and Qinke Wang. Implementation and extensibility of an analytic placer. In *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 2005.

[54] Andrew A. Kennings and Igor L. Markov. Analytical minimization of half-perimeter wirelength. In *Proc. Asia and South Pacific Design Automation Conf.*, pages 179–184, 2000.

[55] J. Kleinhans, G. Sigl, F. M. Johannes, and K. Antreich. GORDIAN: VLSI placement by quadratic programming and slicing optimization. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, CAD-10:356–365, March 1991.

[56] Tim (Tianming) Kong. A novel net weighting algorithm for timing-driven placement. In *Proc. Int. Conf. on Computer Aided Design*, pages 172–176, 2002.

[57] Frank M. Johannes Konrad Doll and Kurt J. Antreich. Iterative placement improvement by network flow methods. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 13(10), 1994.

[58] Tao Luo, David Newmark, and David Z. Pan. A new LP based incremental timing driven placement for high performance designs. In *Proc. Design Automation Conf.*, 2006.

[59] Tao Luo and David Z. Pan. Large scale placement with explicit cell movement control. In *Technical Report UT-CERC-06-01*, April 2006.

[60] Tao Luo, Haoxing Ren, Charles J. Alpert, and David Z. Pan. Computational geometry based placement migration. In *Proc. Int. Conf. on Computer Aided Design*, 2005.

[61] M. Marek-Sadowska and S. Lin. Timing driven placement. pages 94–97, 1989.

[62] TimberWolf Systems, Inc.. Timberwolf placement & global routing software package. In http://www2.twolf.com/benchmark.html.

[63] Fan Mo, Abdallah Tabbara, and Robert K. Brayton. A force-directed macro-cell placer. In *Proc. Int. Conf. on Computer Aided Design*, 2000.

[64] MOSEK. http://www.mosek.com. 2005.

[65] Gi-Joon Nam. ISPD 2006 placement contest: Benchmark suite and results. In *Proc. Int. Symp. on Physical Design*, 2006.

[66] Gi-Joon Nam, Charles J. Alpert, Paul Villarrubia, Bruce Winter, and Mehmet Yildiz. The ISPD2005 placement contest and benchmark suite. In *Proc. Int. Symp. on Physical Design*, 2005.

[67] W. C. Naylor, R. Donelly, and L. Sha". Non-linear optimization system and method for wire length and dealy optimization for an automatic electric circuit placer. US patent 6,301,693, 2001.

[68] OAGear:. http://openedatools.si2.org/oagear/.

[69] OpenAccess. http://openeda.si2.org/.

[70] R.H.J.M. Otten. Global wires harmful? In *Proc. Int. Symp. on Physical Design*, pages 104–109, Apr. 1998.

[71] P. Cocchini P. Saxena, N. Menezes and D. A. Kirkpatrick. Repeater scaling and its impact on cad. In *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 2004.

[72] Min Pan, N. Viswanathan, and C. Chu. An efficient and effective detailed placement algorithm. In *Proc. Int. Conf. on Computer Aided Design*, 2005.

[73] Haifeng Qian and Sachin S. Sapatnekar. A hybrid linear equation solver and its application in quadratic placement. In *Proc. Int. Conf. on Computer Aided Design*, 2005.

[74] H. Ren, D. Z. Pan, and P. Villarrubia. True crosstalk aware incremental placement with noise map. In *Proc. Int. Conf. on Computer Aided Design*, pages 616–619, 2004.

[75] Haoxing Ren, David Z. Pan, Charles J. Alpert, and P. Villarrubia. Diffusion-based placement migration. In *Proc. Design Automation Conf.*, June, 2005.

[76] Haoxing Ren, David Zhigang Pan, and David S. Kung. Sensitivity guided net weighting for placement driven synthesis. In *Proc. Int. Symp. on Physical Design*, pages 10–17, 2004.

[77] Jarrod Roy, Saurabh Adya, David Papa, and Igor Markov. Min-cut floorplacement. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 2006.

[78] Sachin S. Sapatnekar and Weitong Chuang. Power-delay optimizations in gate sizing. *ACM Trans. Des. Autom. Electron. Syst.*, 5(1):98–114, 2000.

[79] Majid Sarrafzadeh, David Knol, and Gustavo Tellez. Unification of budgeting and placement. In *Proc. Design Automation Conf.*, pages 758–761, 1997.

[80] G. Sigl, K. Doll, and F. M. Johannes. Analytical placement: A linear or quadratic objective function? In *Proc. Design Automation Conf.*, pages 427–432, 1991.

[81] Peter Spindler and Frank M. Johannes. Fast and robust quadratic placement combined with an exact linear net model. In *Proc. Int. Conf. on Computer Aided Design*, 2006.

[82] A. Srinivasan, K. Chaudhary, and E. S. Kuh. Ritual: A performance driven placement algorithm for small cell ICs. In *Proc. Int. Conf. on Computer Aided Design*, pages 48–51, 1991.

[83] William Swartz and Carl Sechen. Timing driven placement for large standard cell circuits. In *Proc. Design Automation Conf.*, pages 211–215, 1995.

[84] N. Viswanathan and C. C. N. Chu. Fastplace: Efficient analytical placement using cell shifting, iterative local refinement and a hybrid net model. In *Proc. Int. Symp. on Physical Design*, pages 26–33, 2004.

[85] N. Viswanathan and C. C. N. Chu. Fastplace 3.0: A fast multilevel quadratic placement algorithm with placement congestion control. In *Proc. Asia and South Pacific Design Automation Conf.*, 2007.

[86] K. Vorwerk, A. Kennings, and A. Vannelli. Engineering details of a stable force-directed placer. In *Proc. Int. Conf. on Computer Aided Design*, 2004.

[87] M. Wang, X. Yang, and M. Sarrafzadeh. Dragon2000: Standard-cell placement tool for large industry circuits. In *Proc. Int. Conf. on Computer Aided Design*, pages 260–263, 2000.

[88] Z. Xiu, J. D. Ma, S. M. Fowler, and R. A. Rutenbar. Large-scale placement by grid-warping. In *Proc. Design Automation Conf.*, pages 351–356, 2004.

[89] Zhong Xiu, David A. Papa, and et.al. Early research experience with OpenAccess gear: an open source development environment for physical design. In *Proc. Int. Symp. on Physical Design*, 2005.

[90] Bo Yao, Hongyu Chen, Chung-Kuan Cheng, Nan-Chi Chou, Lung-Tien Liu, and Peter Suaris. Unified quadratic programming approach for mixed mode placement. In *Proc. Int. Symp. on Physical Design*, 2005.

[91] Mehmet Can Yildiz and Patrick H. Madden. Improved cut sequences for partitioning based placement. In *Proc. Design Automation Conf.*, 2001.

Vita

Tao Luo was born in Lanzhou, Gansu province, China. He received a Bachelor's degree in Electro-Mechanical Engineering from the University of Electronic Science and Technology of China, Chengdu, China, in 1995, a Master's degree in Management Information System from TongJi University, Shanghai, China, in 1999, and a Master's degree in Computer Engineering from the University of Texas at Austin, Austin, Texas, in 2004. Tao Luo worked as a part-time software tools engineer at StarCore, LLC. from May 2003 to May 2004. He worked as a co-op in the design verification group at Analog Devices, Inc. from June to December 2004. He worked as an intern for VLSI CAD research at AMD North Austin Design Center, from May 2005 to October 2005 and May 2006 to December 2006. He worked as an intern for VLSI CAD research at IBM Austin Research Laboratory during the Summer of 2007. He received the Engineering Doctoral Fellowship from the University of Texas at Austin in 2006 and 2007, and he received the best paper in session award in SRC TECHON conference 2007.

Permanent address: 2501 Lake Austin Blvd. Apt. C101
Austin, Texas 78703

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.